

Android经典项目案例开发实战宝典

11小时高清多媒体教学视频

郭金尚　等编著



资深Android程序员用心之作，多年开发经验毫无保留，以飨读者
立足实战，通过20个经典项目案例，全面、深入地阐释了Android开发的精髓

- **注重实战：** 详解20个Android经典项目案例的开发过程，提高实战开发水平
- **内容全面：** 涵盖Android本地开发、网络开发、多媒体影音开发和游戏开发等领域
- **由浅入深：** 从较为简单的案例开始，逐渐加大难度，适合各个层次的读者阅读
- **技巧丰富：** 给出了大量的开发技巧，攻克各种疑点和难点，迅速提高开发水平
- **代码经典：** 每个案例都给出了详细的源代码，并提供了大量的注释，便于读者研读
- **贴心专栏：** 每个案例后都专门设有特色栏目“知识拓展”，以拓宽读者的知识面
- **视频教学：** 每章都提供了高清多媒体教学视频，便于读者更加轻松、直观地学习

清华大学出版社

Android
经典项目案例开发实战宝典

郭金尚　等编著

清华大学出版社

内 容 简 介

本书详细介绍了20个Android项目案例的实际开发过程，以提升读者的实际项目开发水平。本书案例紧贴市场，讲解由浅入深，并注重对实际动手能力的提高，还配以翔实的开发情景截图，同时还将重要的知识点、开发技巧以“知识拓展”的形式呈现给读者。另外，作者专门为本书录制了大量的配套教学视频，以帮助读者更好地学习本书内容，这些视频和书中的实例源代码一起收录于本书的配书光盘中。

全书共分5篇。第1篇介绍Android开发环境及搭建、Android工程的创建和调试方法；第2篇介绍计算器、电子词典、文件管理器、备忘录、短信收发工具、通讯录、任务管理器、软件管理器；第3篇介绍Android公交查询、股票查询软件、Google天气客户端、RSS新闻阅读器、Android地图应用、新浪微博客户端；第4篇介绍MP3播放器、Android照相机、视频播放器；第5篇介绍小兔跳铃铛、飞行射击游戏、3D迷宫游戏。

本书涉及面广，从应用到游戏，从简单到复杂，几乎涉及Android项目开发的所有类型。本书适合所有想全面学习Android开发技术的人员阅读，也适合各种使用Android进行开发的工程技术人员使用。对于经常使用Android做开发的人员，本书更是一本不可多得的案头必备参考书。

图书在版编目（CIP）数据

Android经典项目案例开发实战宝典 / 郭金尚等编著. —北京：清华大学出版社，2013
ISBN 978-7-302-32101-9

Ⅰ. ①A…　Ⅱ. ①郭…　Ⅲ. ①移动电话机—应用软件　Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字（2013）第082727号

责任编辑：夏兆彦
封面设计：欧振旭
责任校对：胡伟民
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**40.75　　**字　　数：**1020千字
版　　次：2013年9月第1版　　**印　　次：**2013年9月第1次印刷
印　　数：1～ 000
定　　价：　　元

产品编号：052481-01

前　　言

从 2008 年 10 月第一部 Android 智能手机发布到 2011 年第一季度，Android 在全球的市场份额首次超过塞班系统，跃居全球第一，短短三年时间 Android 可以说是称霸全球。2012 年 11 月数据显示，Android 占据全球智能手机操作系统市场 76%的份额，中国市场占有率为 90%，如此庞大的用户群体，相信没有一个开发者不会为之心动。

由于 Android 发展迅速，导致了就业市场对 Android 开发人员的需求量猛增。然而，很多企业需要的是拥有实践经验的开发人员。刚毕业的大学生一般没有企业要求的实践经验，而培训机构的高昂培训费又令他们望而却步。尽管可以通过很多 Android 书籍中的小例子积累一些经验，但这些例子毕竟有限，有的也不完整，根本达不到企业所要求的水平。

为了帮助读者提高 Android 实际应用开发水平，笔者结合自己多年的开发经验和心得体会，花费近一年的时间写作了本书。希望各位读者能在本书的引领下提高 Android 实际项目开发水平，成为一名开发高手。

本书详细介绍了 20 个 Android 项目案例的开发过程，这些案例紧贴市场，实用价值高，读者稍加修改便可用于自己的项目当中。为了便于读者高效而直观地阅读本书内容，笔者还专门为本书录制了大量的配套教学视频。学习完本书后，读者应该可以具备独立进行项目开发的能力。

本书特色

1．配备大量多媒体语音教学视频，学习效果好

作者专门录制了大量的配套多媒体语音教学视频，以便让读者更加轻松、直观地学习本书内容，提高学习效率。这些视频与本书源代码一起收录于配书光盘中。

2．内容全面、系统、深入

本书系统介绍了 Android 开发中常见的几种项目类型，包括本地应用开发、网络应用开发、多媒体影音开发及游戏开发，提供了共计 20 个项目案例。这些项目案例可以让读者在实际操作中掌握开发流程，巩固基础知识，攻克难点疑点。

3．讲解由浅入深，循序渐进

本书从 Android 搭建环境讲起，前面介绍较为简单的案例，到后面逐渐加大难度，让读者能循序渐进，更好地接受本书的内容。

4．贯穿大量的开发实例和技巧，迅速提升开发水平

本书以大量的典型实例贯穿全文，并给出了大量的开发技巧，以便让读者更好地理解

各种概念和开发技术，体验实际编程，迅速提高开发水平。

5．详解典型项目案例开发，提高实战水平

本书的每一个案例都给出了详细的开发过程，并配以大量而详细的代码及注释，让不同层次的读者均可以较好接受，对整个项目学习更全面。

6．提供技术支持，答疑解惑

读者在阅读本书时有任何疑问，都可以发 E-mail 到 guojinshangb@163.com 或者 bookservice2008@163.com 以获得帮助。

本书内容及体系结构

第1篇　Android起步技术（第1～2章）

本篇主要内容包括：Android 开发环境的搭建、Android 基本应用程序的创建、Eclipse 的基本使用等。通过本篇的学习，读者可以掌握如何搭建 Android 开发环境和如何新建一个 Android 项目。

第2篇　Android典型应用实战案例（第3～10章）

本篇主要内容包括：计算器、电子词典、文件管理器、备忘录、短信收发工具、通讯录、任务管理器、软件管理器等。通过本篇的学习，读者可以掌握 Android 应用编程的方法和思路。

第3篇　Android网络应用实战案例（第11～16章）

本篇主要内容包括：Android 公交查询、Android 股票软件、Google 天气客户端、RSS 新闻阅读器、Android 地图应用、新浪微博客户端等。通过本篇的学习，读者可以掌握 Android 网络编程技术。

第4篇　Android影音应用实战案例（第17～19章）

本篇主要内容包括：MP3 播放工具、Android 照相机、视频播放器等。通过本篇的学习，读者可以掌握 Android 多媒体应用的编程。

第5篇　Android游戏开发实战案例（第20～22章）

本篇主要内容包括：小兔跳铃铛、飞行射击游戏、3D 迷宫游戏等。通过本篇的学习，读者可以掌握 Android 游戏编程的知识，独立开发设计 Android 游戏。

本书读者对象

- Android 初学者；
- 想全面学习 Android 开发技术的人员；
- Android 专业开发人员；

- 利用 Android 做开发的工程技术人员；
- Android 开发爱好者；
- 大中专院校的学生；
- 社会培训班学员；
- 需要一本案头必备手册的程序员。

本书作者

本书由郭金尚主笔编写。其他参与编写的人员有陈晓建、陈振东、程凯、池建、崔久、崔莎、邓凤霞、邓伟杰、董建中、耿璐、韩红轲、胡超、黄格力、黄缙华、姜晓丽、李学军、刘娣、刘刚、刘宁、刘艳梅、刘志刚、司其军、滕川、王连心、沃怀凯、闫玉宝。

虽然笔者对本书中所述内容都尽量核实，并多次进行文字校对，但因时间所限，可能还存在疏漏和不足之处，恳请读者批评指正。

编者著

目　　录

第 1 篇　Android 起步技术

第 2 篇　Android 典型应用实战案例

第 3 篇 Android 网络应用实战案例

第 4 篇 Android 影音应用实战案例

第 1 篇　Android 起步技术

第 1 章　搭建 Android 开发环境

Android 是基于 Linux 内核的软件平台和操作系统，是 Google 在 2007 年 11 月 5 日公布的手机系统平台，早期由 Google 开发，后由开放手持设备联盟（Open Handset Alliance）开发。

1.1　Android 的诞生

Android 操作系统最初由 Andy Rubin 开发，刚开始主要支持手机，被 Google 收购后，对 Android 进行了改良，使其可以使用于平板电脑等其他领域。

1.1.1　Android 的发展史

Android 的发展历史如下：

- 2003 年 10 月，Andy Rubin 等人创建 Android 公司，并组建 Android 团队。
- 2005 年 8 月 17 日，Google 低调收购了成立仅 22 个月的高科技企业 Android 及其团队。Andy Rubin 成为 Google 公司工程部副总裁，继续负责 Android 项目。
- 2007 年 11 月 5 日，Google 公司正式向外界展示了这款名为 Android 的操作系统，并且在这天 Google 宣布建立一个全球性的联盟组织，该组织由 34 家手机制造商、软件开发商、电信运营商以及芯片制造商共同组成，并与 84 家硬件制造商、软件开发商及电信运营商组成开放手持设备联盟（Open Handset Alliance）来共同研发改良 Android 系统，这一联盟将支持 Google 发布的手机操作系统以及应用软件，Google 以 Apache 免费开源许可证的授权方式，发布了 Android 的源代码。
- 2008 年，在 Google I/O 大会上，Google 提出了 Android HAL 架构图，在同年 8 月 18 日，Android 获得了美国联邦通信委员会（FCC）的批准，在 2008 年 9 月，Google 正式发布了 Android 1.0 系统，这也是 Android 系统最早的版本。
- 2009 年 4 月，Google 正式推出了 Android 1.5 这款手机。从 Android 1.5 版本开始，Google 开始将 Android 的版本以甜品的名字命名，Android 1.5 命名为 Cupcake（纸杯蛋糕）。该系统与 Android 1.0 相比有了很大的改进。
- 2009 年 9 月，Google 发布了 Android 1.6 正式版，并且推出了搭载 Android 1.6 正式版的手机 HTC Hero（G3）。凭借着出色的外观设计以及全新的 Android 1.6 操作系统，HTC Hero（G3）成为当时全球最受欢迎的手机。Android 1.6 也有一个有趣的甜品名称，它被称为 Donut（甜甜圈）。
- 2010 年 2 月，Linux 内核开发者 Greg Kroah-Hartman 将 Android 的驱动程序从 Linux

内核“状态树”（staging tree）上除去，从此，Android 与 Linux 开发主流分道扬镳。在同年 5 月，Google 正式发布了 Android 2.2 操作系统。Google 将 Android 2.2 操作系统命名为 Froyo，翻译成中文名为冻酸奶。

- 2010 年 10 月，Google 宣布 Android 系统达到了第一个里程碑，即电子市场上获得官方数字认证的 Android 应用数量已经达到了 10 万个。Android 系统的应用增长非常迅速，在 2010 年 12 月，Google 正式发布了 Android 2.3 操作系统 Gingerbread（姜饼）。
- 2011 年 1 月，Google 称每日的 Android 设备新用户数量达到了 30 万部，到 2011 年 7 月，这个数字增长到 55 万部，而 Android 系统设备的用户总数达到了 1.35 亿，Android 系统已经成为智能手机领域占有量最高的系统。
- 2011 年 8 月 2 日，Android 手机已占据全球智能手机市场 48%的份额，并在亚太地区市场占据统治地位，终结了 Symbian（塞班系统）的霸主地位，跃居全球第一。
- 2011 年 9 月，Android 系统的应用数目已经达到了 48 万，而在智能手机市场，Android 系统的占有率则达到了 43%，继续排在移动操作系统首位。在 9 月 19 日，Google 发布全新的 Android 4.0 操作系统，这款系统被 Google 命名为 Ice Cream Sandwich（冰激凌三明治）。
- 2012 年 1 月 6 日，Google Android Market 已有 10 万开发者推出超过 40 万活跃的应用，大多数的应用程序为免费。Android Market 应用程序商店目录在新年首周周末突破 40 万基准，距离突破 30 万应用仅 4 个月。在 2011 年早些时候，Android Market 从 20 万增加到 30 万应用也花了 4 个月。

1.1.2　Android 的发行版本

Android 在正式发行之前，最开始拥有两个内部测试版本，并且以著名的机器人名称来对其进行命名，它们分别是：阿童木（Android Beta）、发条机器人（Android 1.0）。后来由于涉及到版权问题，Google 将其命名规则变更为用甜点作为它们系统版本的代号，如图 1.1 所示。甜点命名法开始于 Android 1.5 发布的时候。作为每个版本代表的甜点的尺寸越变越大，然后按照 26 个字母顺序：纸杯蛋糕（Android 1.5），甜甜圈（Android 1.6），松饼（Android 2.0/2.1），冻酸奶（Android 2.2），姜饼（Android 2.3），蜂巢（Android 3.0），冰激凌三明治（Android 4.0），根据最新消息，新一代 Android 版本将命名为果冻豆（Jelly Bean）。



图 1.1　Android 发行版本

Android 各个发行版本及更新如下：

- Android 1.1：2008 年 9 月发布的 Android 第一版。
- Android 1.5 Cupcake（纸杯蛋糕）：2009 年 4 月 30 日发布。主要的更新如下：

拍摄/播放影片，并支持上传到 Youtube；支持立体声蓝牙耳机，同时改善自动配对性能；最新的采用 WebKit 技术的浏览器，支持复制/粘贴和页面中搜索；GPS 性能大大提高；提供屏幕虚拟键盘；主屏幕增加音乐播放器和相框 widgets；应用程序自动随着手机旋转；短信、Gmail、日历、浏览器的用户接口大幅改进，如 Gmail 可以批量删除邮件；相机启动速度加快，拍摄图片可以直接上传到 Picasa；来电照片显示。

- Android 1.6 Donut（甜甜圈）：2009 年 9 月 15 日发布。主要的更新如下：

重新设计的 Android Market 手势；支持 CDMA 网络；文字转语音系统（Text-to-Speech）；快速搜索框；全新的拍照接口；查看应用程序耗电；支持虚拟私人网络（VPN）；支持更多的屏幕分辨率；支持 OpenCore 2 媒体引擎；新增面向视觉或听觉困难人群的易用性插件。

- Android 2.0/2.0.1/2.1 Eclair（松饼）：2009 年 10 月 26 日发布。主要的更新如下：

优化硬件速度；Car Home 程序；支持更多的屏幕分辨率；改良的用户界面；新的浏览器的用户接口和支持 HTML 5；新的联系人名单；更好的白色/黑色背景比率；改进 Google Maps 3.1.2；支持 Microsoft Exchange；支持内置相机闪光灯；支持数码变焦；改进的虚拟键盘；支持蓝牙 2.1；支持动态桌面的设计。

- Android 2.2/2.2.1 Froyo（冻酸奶）：2010 年 5 月 20 日发布。主要的更新如下：

整体性能大幅度提升；3G 网络共享功能；Flash 的支持；App2sd 功能；全新的软件商店；更多的 Web 应用 API 接口的开发。

- Android 2.3.x Gingerbread（姜饼）：2010 年 12 月 7 日发布。主要的更新如下：

增加了新的垃圾回收和优化处理事件；原生代码可直接存取输入和感应器事件、EGL/OpenGL ES、OpenSL ES；新的管理窗口和生命周期的框架；支持 VP8 和 WebM 视频格式，提供 AAC 和 AMR 宽频编码，提供了新的音频效果器；支持前置摄像头、SIP/VOIP 和 NFC（近场通讯）；简化界面、速度提升；更快更直观的文字输入；一键文字选择和复制/粘贴；改进的电源管理系统；新的应用管理方式。

- Android 3.0 Honeycomb（蜂巢）：2011 年 2 月 2 日发布。主要更新如下：

针对平板优化；全新设计的 UI 增强网页浏览功能；n-app purchases 功能。

- Android 3.1 Honeycomb（蜂巢）：2011 年 5 月 11 日发布。版本主要更新如下：

经过优化的 Gmail 电子邮箱；全面支持 Google Maps；将 Android 手机系统与平板系统再次合并从而方便开发者；任务管理器可滚动，支持 USB 输入设备（键盘、鼠标等）；支持 Google TV，可以支持 XBOX 360 无线手柄；widget 支持的变化，能更加容易地定制屏幕 widget 插件。

- Android 3.2 Honeycomb（蜂巢）：2011 年 7 月 13 日发布。版本更新如下：

支持 7 英寸设备；引入了应用显示缩放功能。

- Android 4.0 Ice Cream Sandwich（冰激凌三明治）：2011 年 10 月 19 日发布。版本主要更新如下：

全新的 UI；全新的 Chrome Lite 浏览器，有离线阅读、16 标签页、隐身浏览模式等；截图功能；更强大的图片编辑功能；自带照片应用堪比 Instagram，可以加滤镜、加相框，

进行 360° 全景拍摄，照片还能根据地点来排序；Gmail 加入手势、离线搜索功能，UI 更强大；新功能 People：以联系人照片为核心，界面偏重滑动而非点击，集成了 Twitter、Linkedin、Google+等通讯工具，有望支持用户自定义添加第三方服务；新增流量管理工具，可具体查看每个应用产生的流量，限制使用流量，到达设置标准后自动断开网络。

- Android 4.1 Jelly Bean（果冻豆）。新特性如下：

更快、更流畅、更灵敏；特效动画的帧速提高至 60fps，增加了三倍缓冲；增强通知栏；全新搜索，搜索带来全新的 UI、智能语音搜索和 Google Now 三项新功能；桌面插件自动调整大小；加强无障碍操作；语言和输入法扩展；新的输入类型和功能；新的连接类型。

1.2　Android 的系统架构及特性

1.2.1　Android 的系统架构

如图 1.2 可以看出，Android 操作系统的体系结构可分为 4 层，由上到下依次是应用程序、应用程序框架、核心类库和 Linux 内核，其中第三层还包括 Android 运行时的环境。下面分别来讲解各个部分。

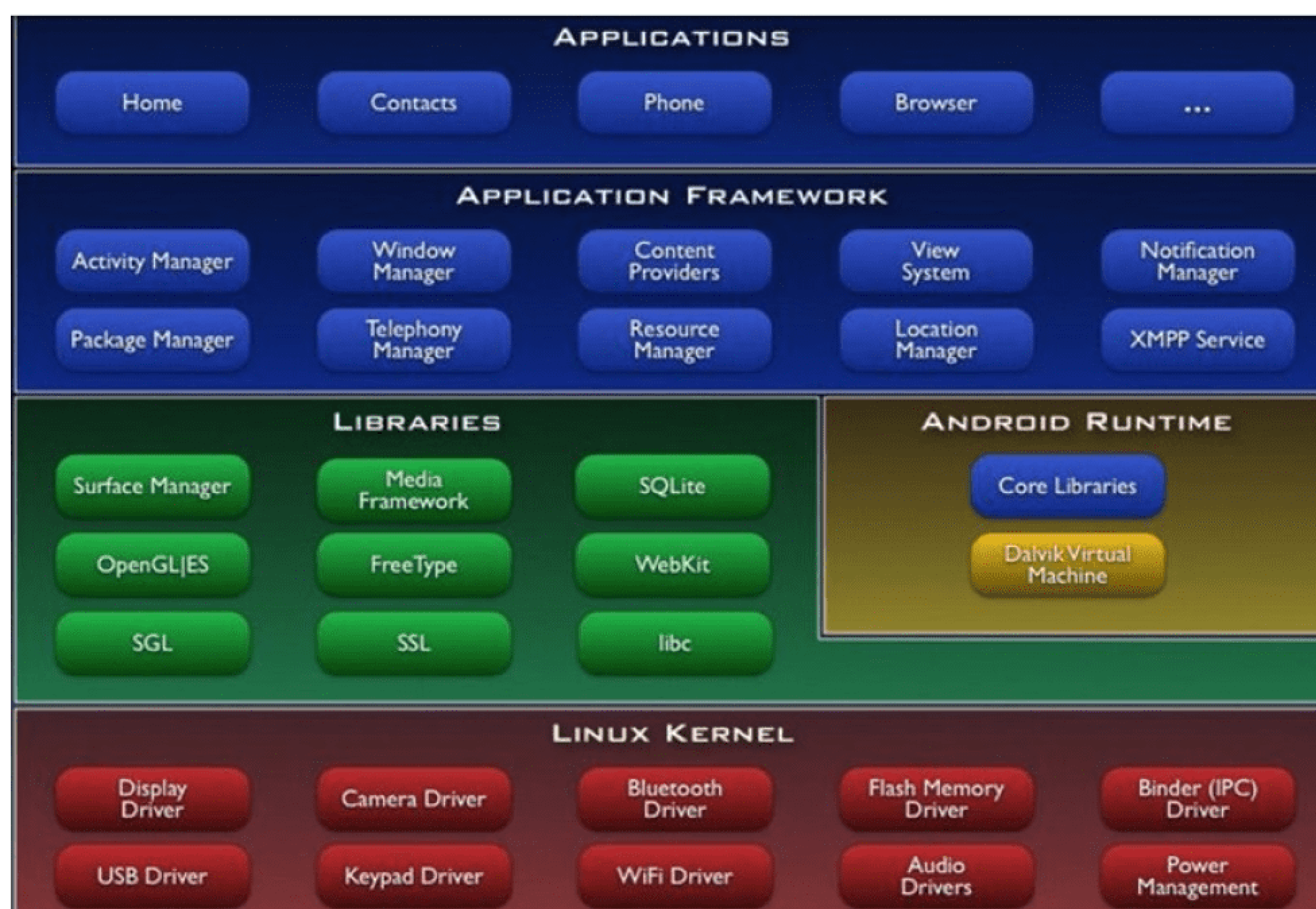


图 1.2　Android 系统架构

1. 应用程序

Android 连同一个核心应用程序包一起发布，该应用程序包包括 E-mail 客户端、SMS 短消息程序、日历、地图、浏览器、联系人管理程序等。所有的应用程序都是用 Java 编写的。

2．应用程序框架

开发者完全可以访问核心应用程序所使用的 API 框架。该应用程序框架架构用来简化组件软件的重用，任何一个应用程序都可以发布它的功能块，并且任何其他的应用程序都可以使用其发布的功能块（不过得遵循框架的安全性限制）。该应用程序重用机制使得组件可以被用户替换。

所有的应用程序都由一系列的服务和系统组成，包括：

（1）一个可扩展的视图（Views），可以用来创建应用程序，包括列表（lists）、网络（grids）、文本框（text boxes）、按钮（buttons），甚至是一个可嵌入的 Web 浏览器。

（2）内容管理器（Content Providers），使得应用程序可以访问另一个应用程序的数据（如联系人数据库），或者共享它们自己的数据。

（3）一个资源管理器（Resource Manager），提供非代码资源的访问，如本地字符串、图形和分层文件（layout files）。

（4）一个通知管理器（Notification Manager），使得应用程序可以在状态栏中显示客户通知信息。

（5）一个活动类管理器（Activity Manager），用来管理应用程序生命周期并提供常用的导航回退功能。

3．Android程序库

Android 包括一个被 Android 系统中各种不同组件所使用的 C/C++集库，该库通过 Android 应用程序框架为开发者提供服务。以下是一些主要的核心库：

（1）系统 C 库：一个从 BSD 继承来的标准 C 系统函数库（libc），专门为基于 Embedded Linux 的设备定制。

（2）媒体库：基于 PacketVideo OpenCORE。该库支持录放，并且可以录制许多流行的音频视频格式以及静态映像文件，包括 MPEG4、H.264、MP3、AAC、JPG、PNG。

（3）Surface Manager：对显示子系统的管理，并且为多个应用程序提供 2D 和 3D 图层的无缝融合。

（4）LibWebCore：一个最新的 Web 浏览器引擎，用来支持 Android 浏览器和一个可嵌入的 Web 视图。

（5）SGL：一个内置的 2D 图形引擎。

（6）3D libraries：基于 OpenGL ES 1.0 API 实现。该库可以使用硬件 3D 加速（如果可用）或者使用高度优化的 3D 软加速。

（7）FreeType：位图（bitmap）和向量（vector）字体显示。

（8）SQLite：一个对于所有应用程序可用、功能强劲的轻型关系型数据库引擎。

4．Android运行库

Android 包括了一个核心库，该核心库提供了 Java 编程语言核心库的大多数功能。

每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。Dalvik 是针对同时高效地运行多个 VM 实现的。Dalvik 虚拟机执行.dex 的 Dalvik 可执行文件，该格式文件针对最小内存使用做了优化。该虚拟机是基于寄存器的，所有的类

都经由 Java 汇编器编译，然后通过 SDK 中的 DX 工具转化成.dex 格式由虚拟机执行。

Dalvik 虚拟机依赖于 Linux 的一些功能，如线程机制和底层内存管理机制。

5. Linux内核

Android 的核心系统服务依赖于 Linux 内核，如安全性、内存管理、进程管理、网络协议栈和驱动模型。Linux 内核也同时作为硬件和软件栈之间的硬件抽象层。

1.2.2　Android 的特性

Android 平台有如下特性：

（1）应用程序框架支持组件的重用与替换。

这样我们可以把系统中不喜欢的应用程序删除，安装我们喜欢的应用程序。

（2）Dalvik 虚拟机专门为移动设备进行了优化。

Android 应用程序将由 Java 编写、编译的类文件，通过 DX 工具转换成一种后缀名为.dex 的文件来执行。Dalvik 虚拟机是基于寄存器的，相对于 Java 虚拟机速度要快很多。

（3）内部集成浏览器基于开源的 WebKit 引擎。

有了内置的浏览器，这将意味着 WAP 应用的时代即将结束，真正的移动互联网时代已经来临，手机就是一台“小电脑”，可以在网上随意遨游。

（4）优化的图形库包括 2D 和 3D 图形库，3D 图形库基于 OpenGL ES 1.0。

强大的图形库给游戏开发带来了福音。在 3G 时代最为重要的应用莫过于手机上网和手机游戏。

（5）SQLite 用作结构化的数据存储。

（6）多媒体支持包括常见的音频、视频和静态映像文件格式，如 MPEG4、H.264、MP3、AAC、AMR、JGP、PNG、GIF。

（7）GSM 电话（依赖于硬件）功能。

（8）蓝牙（Bluetooth）、EDGE、3G、WiFi（依赖于硬件）功能。

（9）照相机、GPS、指南针和加速度计（依赖于硬件）功能。

（10）丰富的开发环境，包括设备模拟器、调试工具、内存及性能分析图表和 Eclipse 集成的开发环境插件。

（11）Google 提供了 Android 开发包 SDK，其中包含了大量的类库和开发工具，以及针对 Eclipse 的可视化开发插件 ADT。

1.3　Android 的开发环境的搭建

在开始 Android 开发之前，首先要做的就是搭建开发环境，主要包括 JDK 的安装、Eclipse 的安装、ADT 的安装、Android SDK 的安装及创建 AVD。

1. java JDK安装

进入网页 http://java.sun.com/javase/downloads/index.jsp，看到如图 1.3 所示的 JDK 下载界面，选择最新的 JDK7。

Java Platform, Standard Edition		
Java SE 7u6 JavaFX 2.2 is now bundled with the JDK on Windows, Mac and Linux x86/x64. We also include bug fixes and enhancements. Learn more **"What Java Do I Need?"** You must have a copy of the **JRE** (Java Runtime Environment) on your system to **run** Java applications and applets. To **develop** Java applications and applets, you need the **JDK** (Java Development Kit), which includes the JRE.	**JDK** DOWNLOAD **JDK 7 Docs** Installation Instructions ReadMe ReleaseNotes Oracle License Java SE Products Third Party Licenses Certified System Configurations	**JRE** DOWNLOAD **JRE 7 Docs** Installation Instructions ReadMe ReleaseNotes Oracle License Java SE Products Third Party Licenses Certified System Configurations
JDK 7 and JavaFX Demos and Samples Demos and samples of common tasks and new functionality available on JDK 7. The source code provided with samples and demos for the JDK is meant to illustrate the usage of a given feature or technique and has been deliberately simplified.	**Demos and Samples** DOWNLOAD	
Java SE 6 Update 34 This release includes tools useful for developing and testing programs written in the Java programming language and running on the JavaTM platform. Learn more	**JDK** DOWNLOAD **JDK 6 Docs**	**JRE** DOWNLOAD **JRE 6 Docs**

图 1.3　JDK 下载

下载完之后直接安装即可。我们检查一下是否已经正确安装。进入 cmd 窗口，输入 java–version，看到如图 1.4 所示的显示，表明 JDK 已经安装正确。

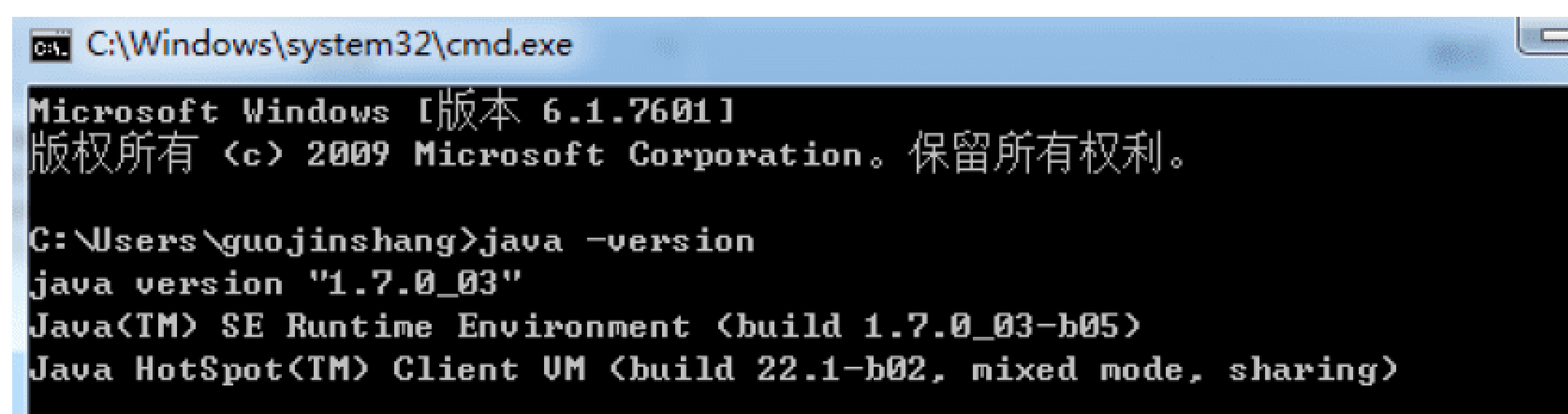


图 1.4　查看系统 Java 版本

2．Eclipse安装

进入网页 http://www.eclipse.org/downloads/，看到 Eclipse 下载界面，如图 1.5 所示。选择第一个 Java EE 下载。

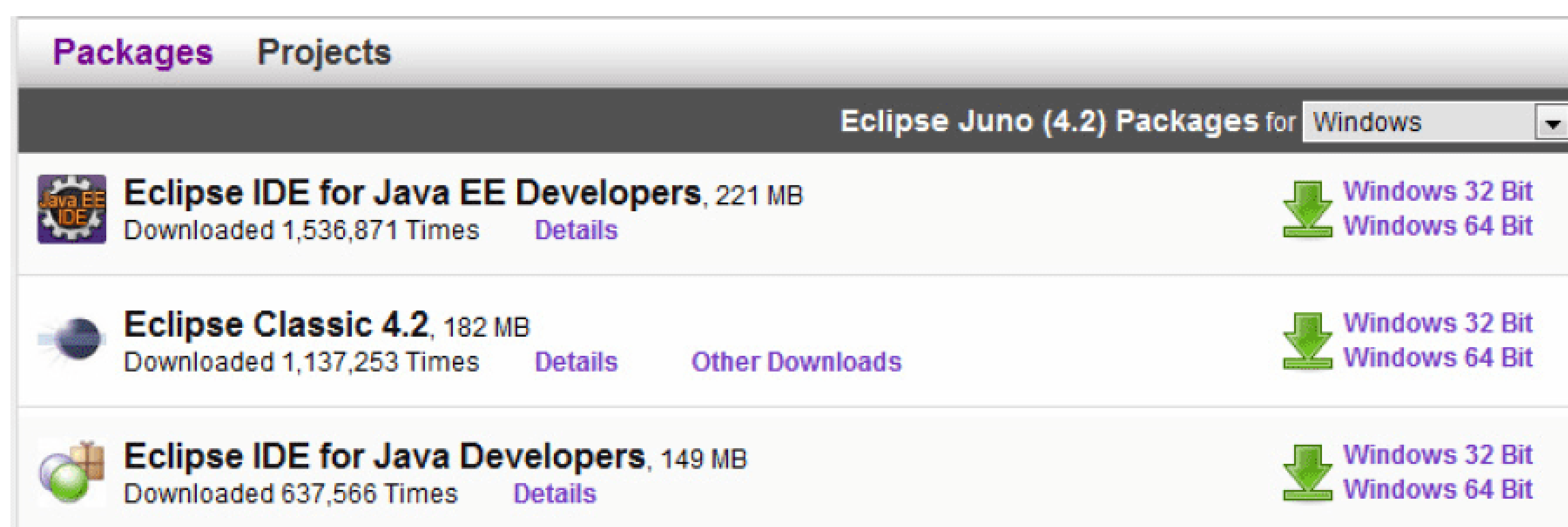


图 1.5　Eclipse 下载

3．ADT 安装

打开 Eclipse IDE，单击 Help|Install New Software 命令，单击 Add...按钮，弹出对话框要求输入 Name 和 Location。Name 自己随便取，Location 输入 http://dl-ssl.google.com/android/eclipse。确定返回后，在 work with 后的下拉列表中选择我们刚才添加的 ADT，我们会看到下面有 Developer Tools，展开它会有 Android DDMS 和 Android Development Tools，勾选它们，如图 1.6 所示。然后按提示一步一步单击 next 按钮，重启 Eclipse 之后可以看到 Eclipse 工具栏多了 Android SDK Manager 和 Android Virtual Device Manager 两个按钮。

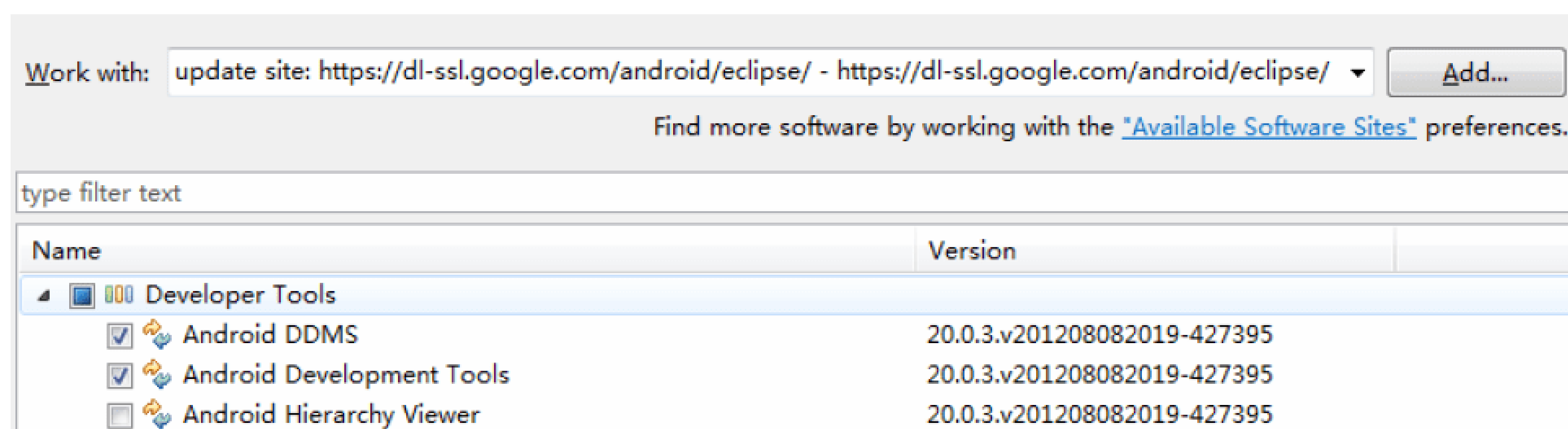


图 1.6　ADT 下载

4．SDK下载

打开 Android SDK Manager，显示当前可用的 SDK 版本，选择你需要的 SDK 版本进行下载，这里我们勾选上 2.3.3、2.3.1 和 2.2 一并下载下来，如图 1.7 所示。当然实际上本书我们只用到 2.3.3 版本，多下载也没什么影响，关键时刻想用一下就方便了。

Android 2.3.3 (API 10)			
SDK Platform	10	1	Installed
Samples for SDK	10	1	Installed
Google APIs	10	2	Installed
Android 2.3.1 (API 9)			
Google APIs	9	2	Installed
EDK	9	1	Installed
Android 2.2 (API 8)			
SDK Platform	8	2	Installed
Samples for SDK	8	1	Installed

图 1.7　SDK 下载

下载完之后，单击 Windows|Preferences 命令，选择 Android，在右侧的 SDK Location 中选择 SDK 安装的位置，单击 OK 按钮即可将 SDK 导入。

5．创建AVD

打开 Android Virtual Device Manager，Name 可以随便设置，Target 选择 API Level 10，SD Card 选择 100M，Skin 选择默认的就行，如图 1.8 所示，最后单击 Create AVD 即可完成创建 AVD。

6．新建一个HelloAndroid程序

单击 File|New |Android Project 命令，建立新项目 HelloAndroid，如图 1.9 所示。

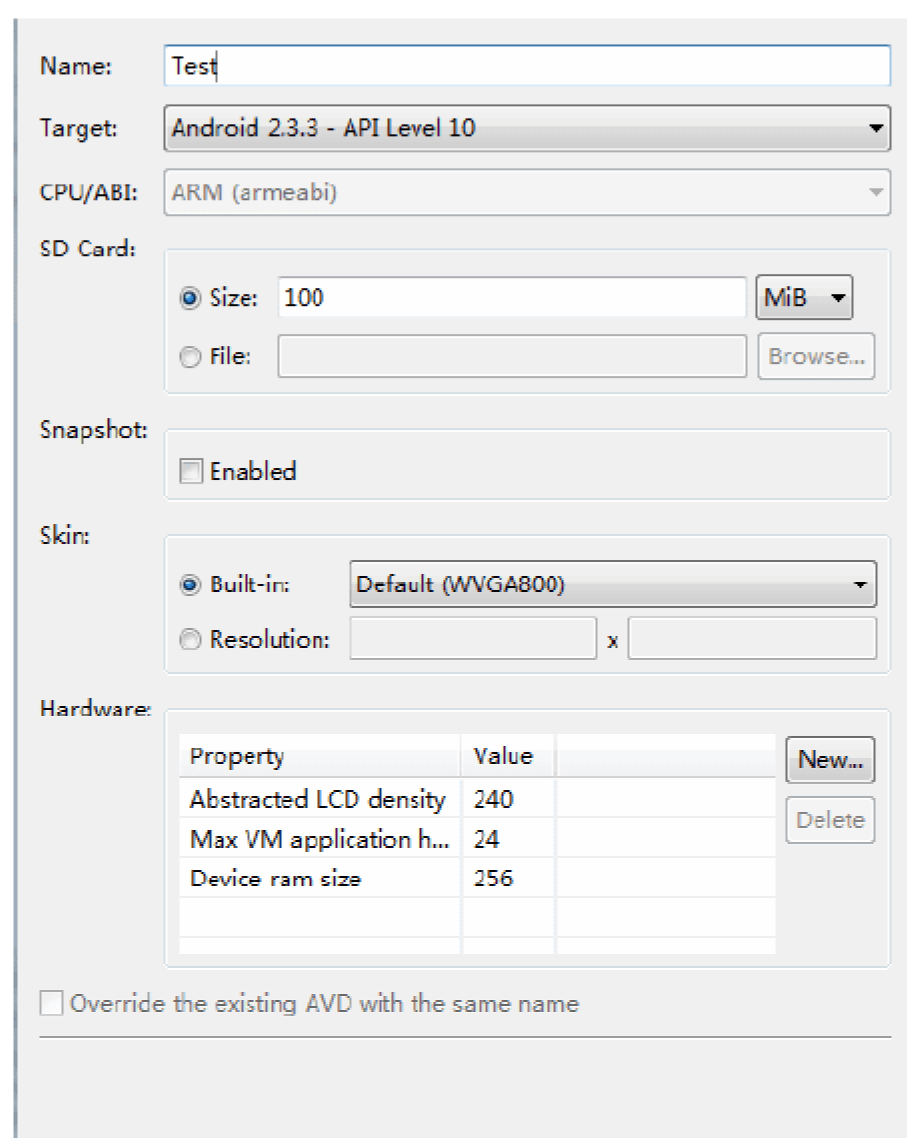


图 1.8　创建 AVD

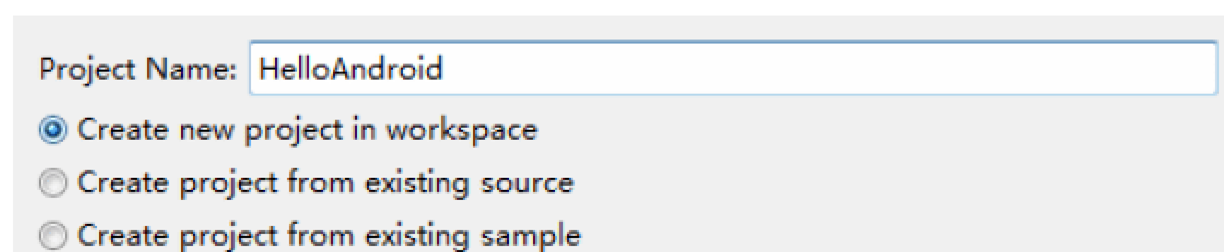


图 1.9　填写项目名称

然后单击 Next 按钮，在 Build Target 中选择“Android 2.3.3”，单击 Next 按钮。填写 Package Name，其他的默认就行，如图 1.10 所示。单击 Finish 按钮。

最后右键单击刚才生成的项目，单击 Run As|Android Application 命令，出现如图 1.11 所示的界面，表明 AVD 启动成功。至此，我们的 Android 开发环境就搭建好了。

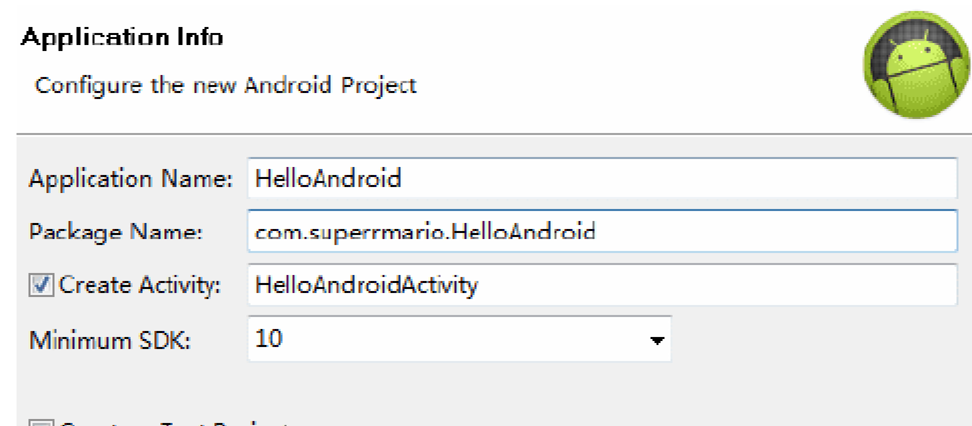


图 1.10　填写包名

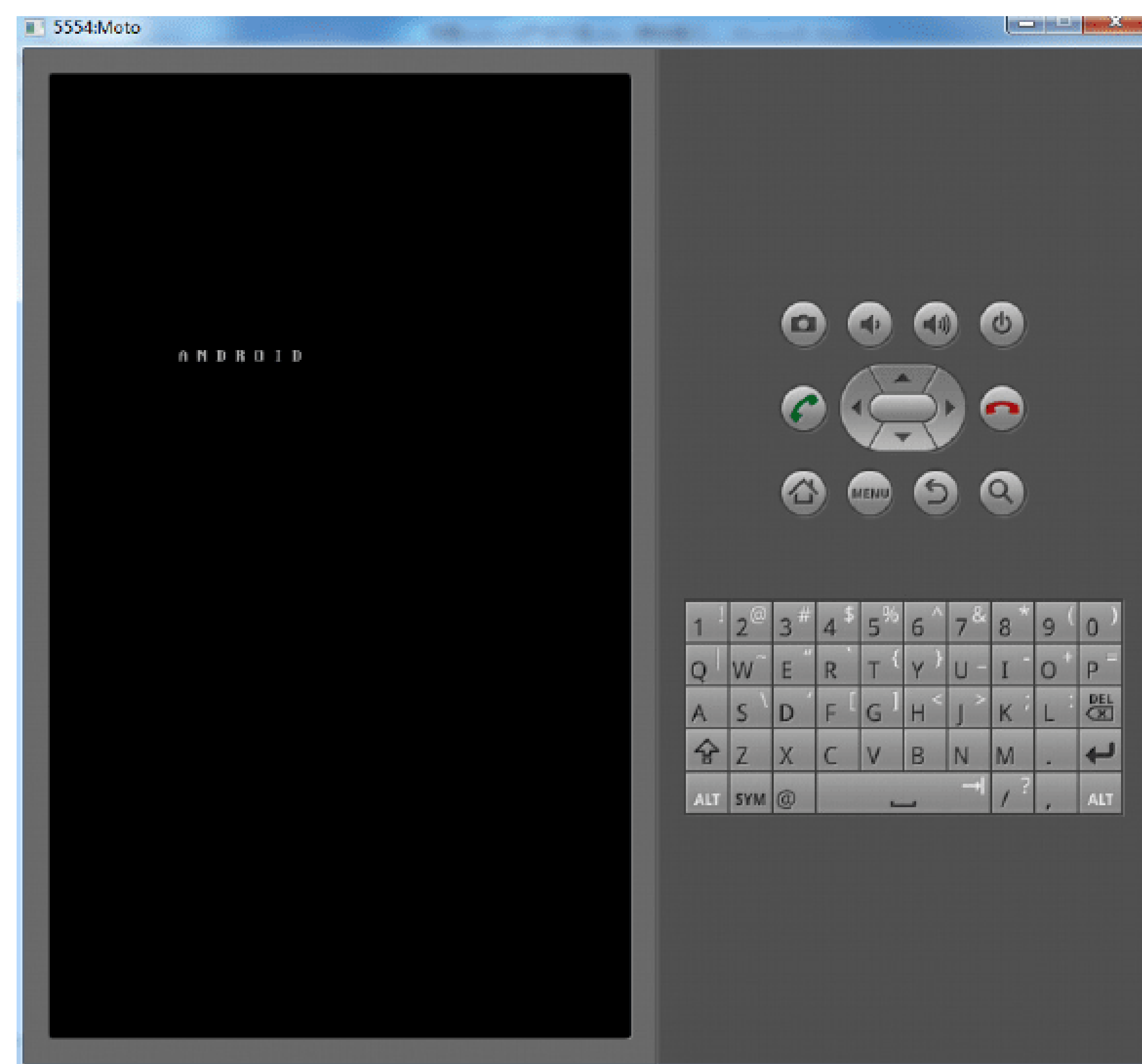


图 1.11　AVD 启动

1.4　Android 调试

Eclipse 为我们提供了方便的调试工具——DDMS，DDMS 的全称是 Dalvik Debug Monitor Service，它为我们提供了多种功能，例如，为测试设备截屏、针对特定的进程查

看正在运行的线程以及堆信息、Logcat、广播状态信息、模拟电话呼叫、接收 SMS、虚拟地理坐标等等。

DDMS 工具存放在{ANDROID-SDK}/tools 路径下，启动 DDMS 的方法如下：

- 直接在 tools 目录下双击 ddms.bat。
- 在 Eclipse 调试程序的过程中启动 DDMS，相应选项在 Eclipse 界面中的位置如图 1.12 所示。

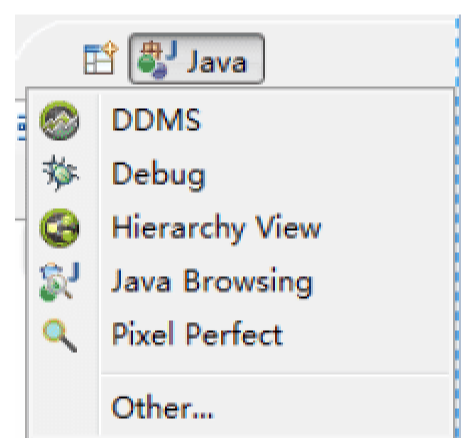


图 1.12　DDMS 工具位置

启动 DDMS 后，界面如图 1.13 所示。

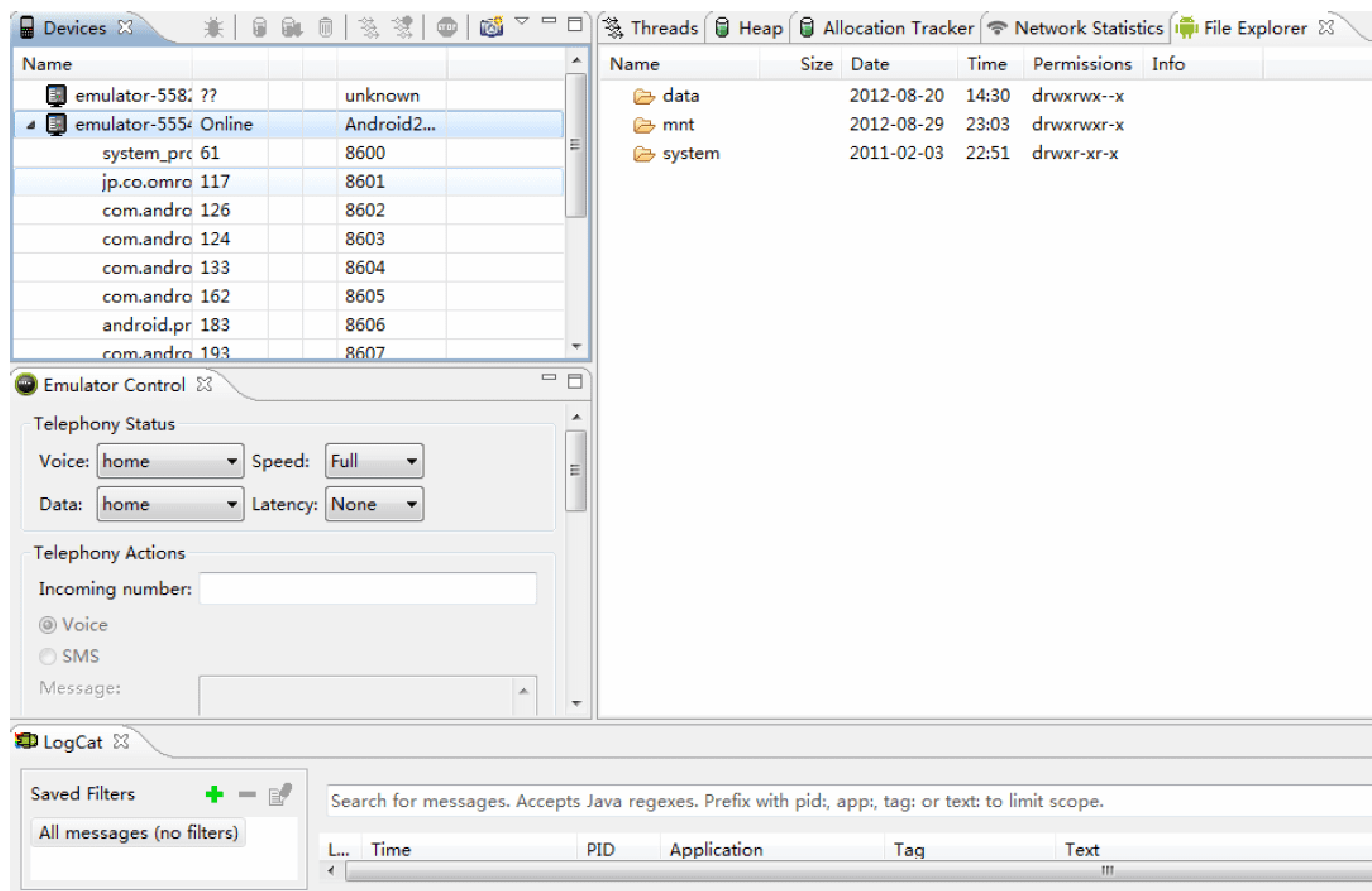


图 1.13　DDMS 界面

DDMS 将搭建起 IDE 与测试终端（Emulator 或者 connected device）的链接，它们应用各自独立的端口监听调试器的信息，DDMS 可以实时监测到测试终端的连接情况。当有新的测试终端连接后，DDMS 将捕捉到终端的 ID，并通过 ADB 建立调试器，从而实现发送指令到测试终端的目的。

如图 1.14 所示，DDMS 可以监听虚拟机中的进程。DDMS 监听的第一个终端 APP 进程的端口为 8600，第二个 APP 进程将分配 8601，如果有更多终端或者更多 APP 进程，将按照这个顺序依次类推。DDMS 通过 8700 端口（base port）接收所有终端的指令。在面板的右上角有一排很重要的按键，它们分别是 Debug the selected process、Update Threads、Update Heap、Stop Process 和 ScreenShot。

在开发过程中，有时需要用到短信、电话功能等，Eclipse 也为我们提供了这些功能的调试工具。通过这个面板的一些功能可以非常容易地使测试终端模拟真实手机所具备的一些交互功能，比如接听电话、根据选项模拟各种不同网络情况、模拟接受 SMS 消息和发送

虚拟地址坐标用于测试 GPS 功能等。下面我们分别来看看模拟打电话和发送短信功能如何操作。

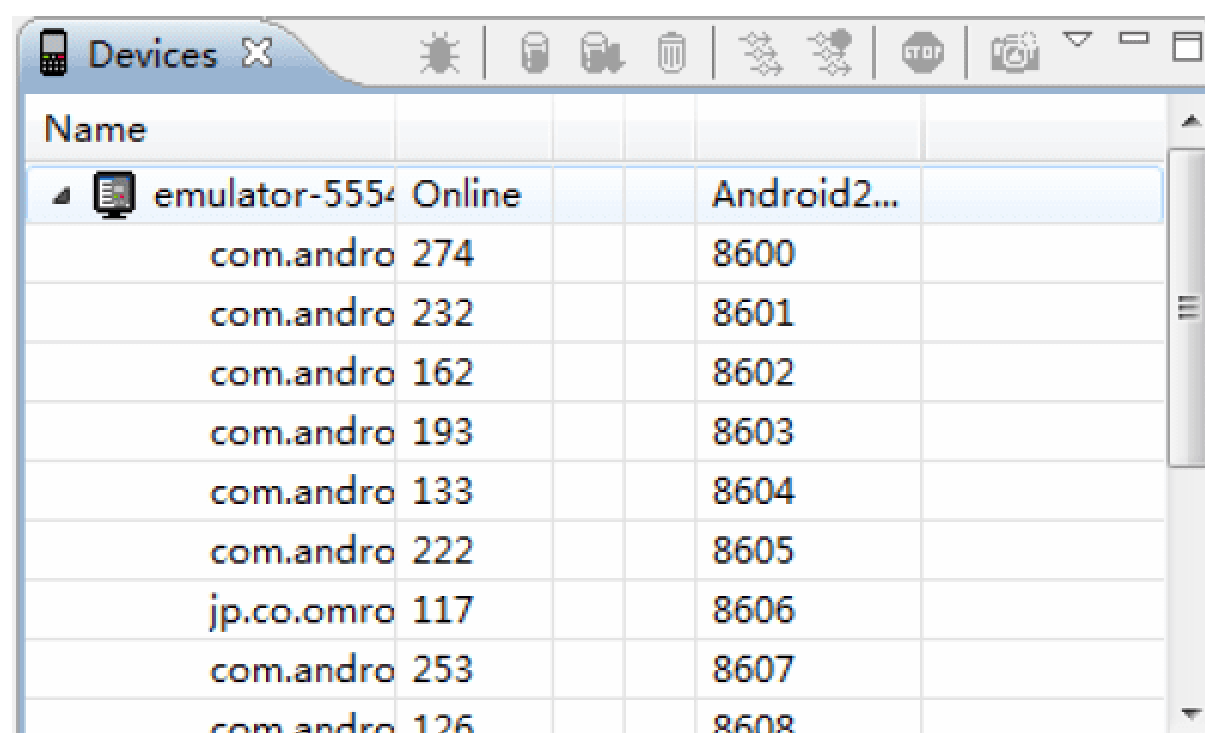


图 1.14　监听 APP 端口

选择要呼叫的模拟器，然后输入呼叫的号码，我们这里输入“12345678”，选择 Voice，单击 Call 按钮，如图 1.15 所示。在模拟器一端就可以看到“12345678”的来电，如图 1.16 所示。

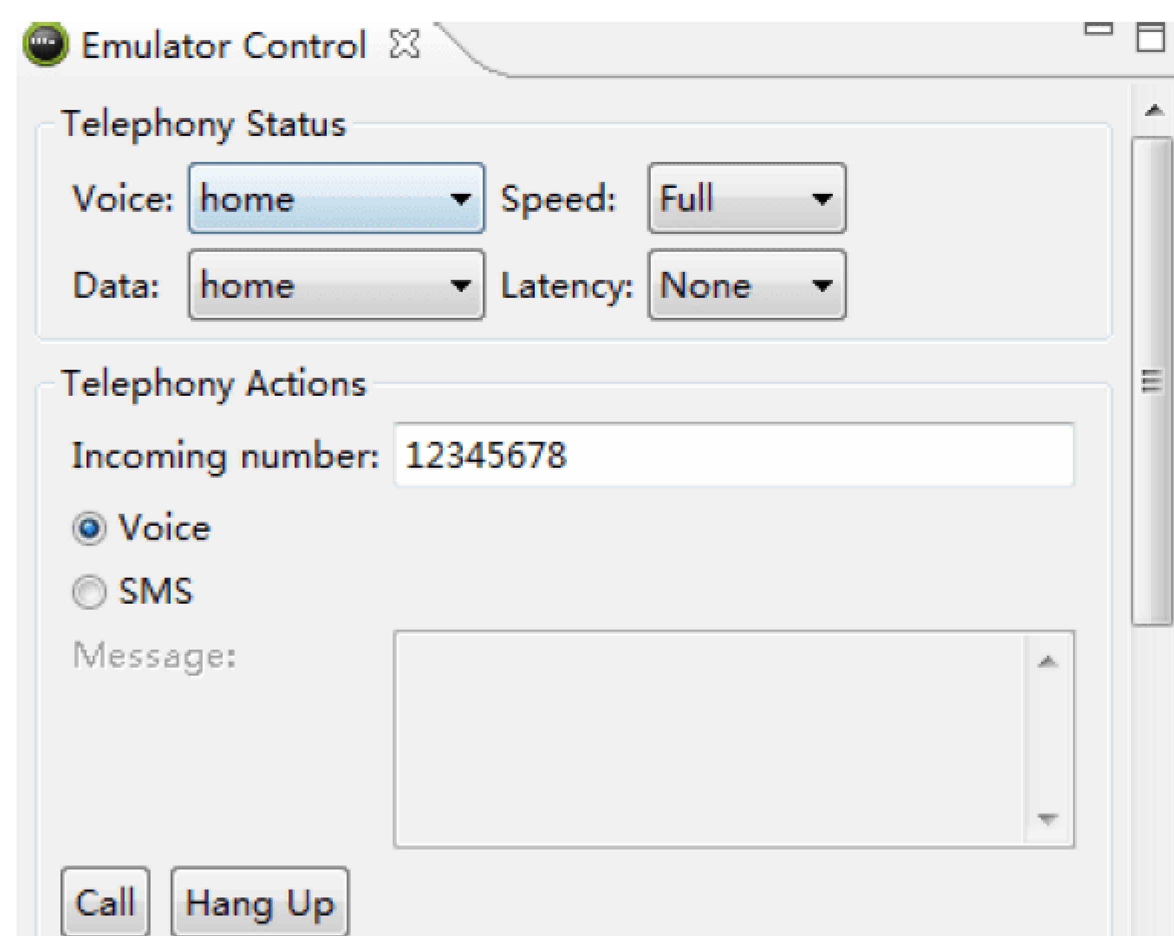


图 1.15　呼叫模拟器

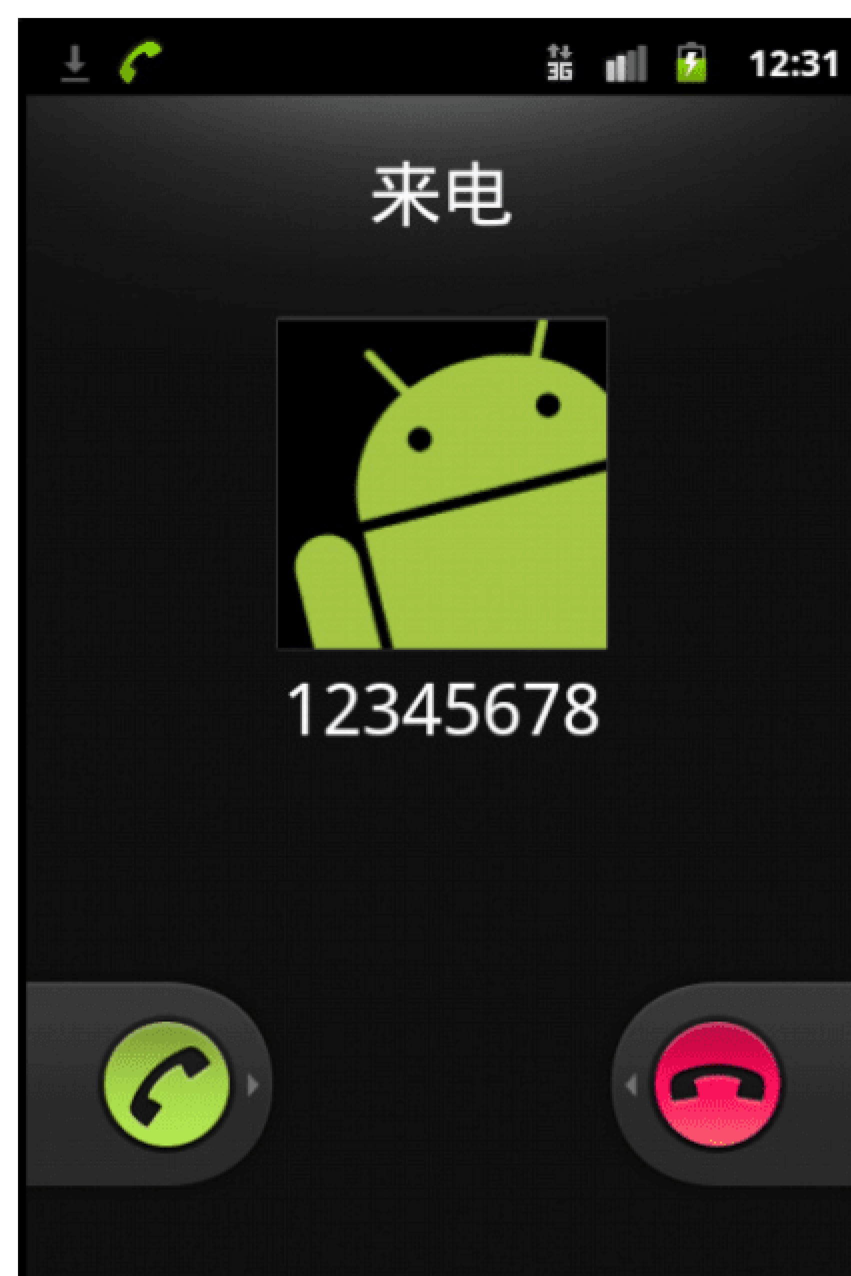


图 1.16　收到来电

发送短信时方法类似，如图 1.17 所示，选择 SMS，输入要发送的信息，单击 Send 按钮。这时在模拟器端可以接收到来自“12345678”的短信“Hello Android!”，如图 1.18 所示。

调试的时候使用最多的要数 logcat 了，logcat 是 Android 的一个命令行工具，可以得到程序的 log 信息。logcat 的功能是由 Android 的类 android.util.Log 决定的，在程序中 log 的使用方法如下所示：

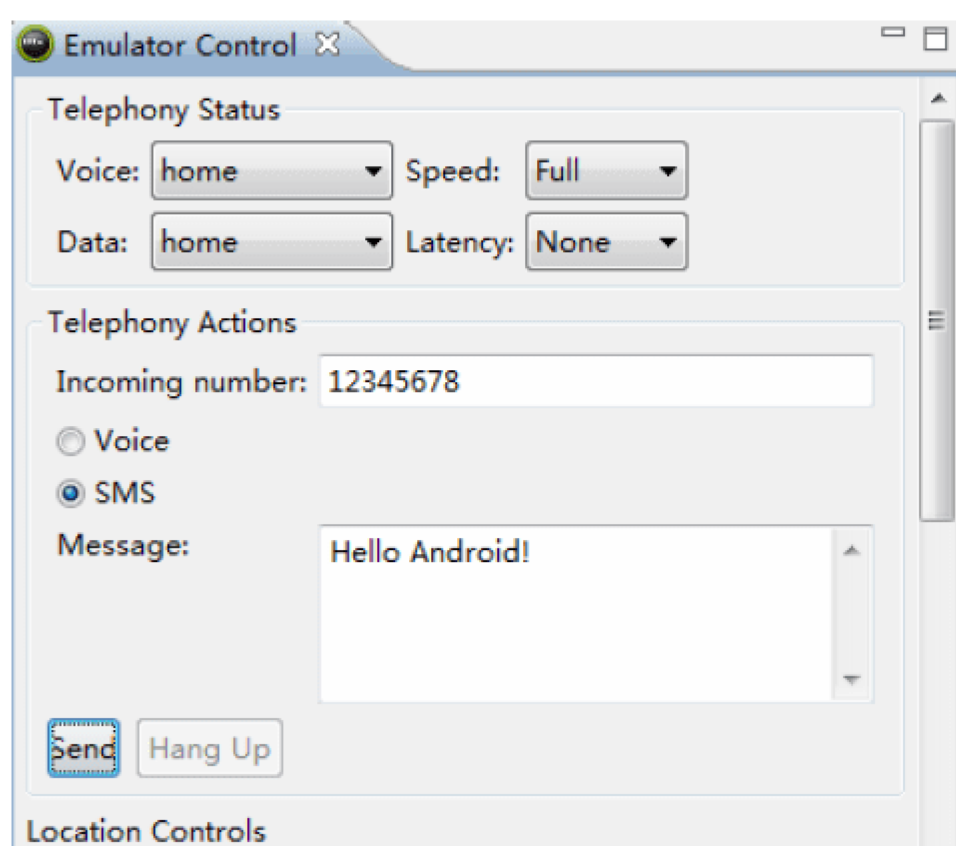


图 1.17　发送短信

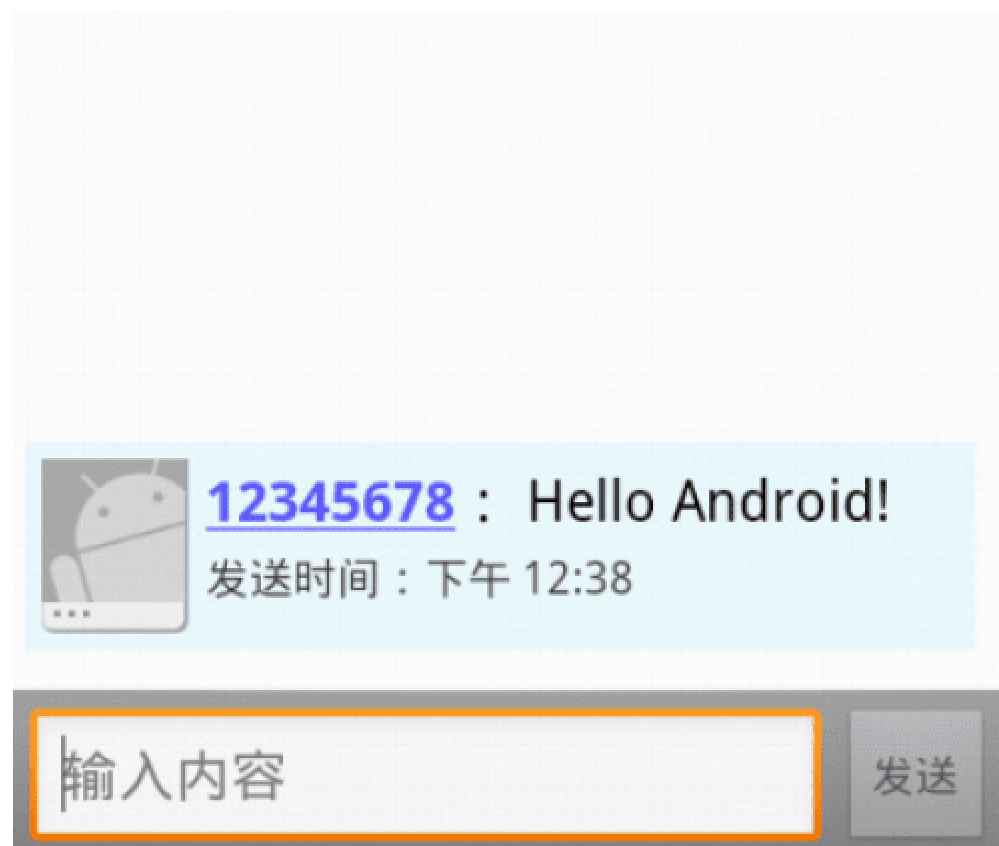


图 1.18　查看短信

```
Log.v() ------------------------- VERBOSE
Log.d() ------------------------- DEBUG
Log.i() ------------------------- INFO
Log.w() ------------------------- WARN
Log.e() ------------------------- ERROR
```

以上 log 的级别依次升高，DEBUG 信息应当只存在于开发中，INFO、WARN、ERROR 这 3 种 log 将出现在发行版中。在使用过程中我们通常声明一个字符串常量 TAG，通过它来区分不同的 log。例如我们在 HelloAndroid 可以定义 private static final String TAG= “HelloAndroid”，这样所有在 HelloAndroid 中使用的 log，均以“HelloAndroid”开头。我们在 HelloAndroid 的 onCreate()函数中打了一个 log——Log.i(TAG,"onCreate!")，运行后可以看到 log 信息，如图 1.19 所示。

L...	Time	PID	Application	Tag	Text
D	08-30 13:59:53.812	2559		AndroidRuntime	>>>>>> AndroidRuntime START com.android.internal.os.RuntimeIn
D	08-30 13:59:53.812	2559		AndroidRuntime	CheckJNI is ON
D	08-30 13:59:54.512	2559		AndroidRuntime	Calling main entry com.android.commands.pm.Pm
D	08-30 13:59:54.542	2559		AndroidRuntime	Shutting down VM
D	08-30 13:59:54.552	2559		dalvikvm	GC_CONCURRENT freed 101K, 71% free 297K/1024K, external 0K/0K
D	08-30 13:59:54.552	2559		dalvikvm	Debugger has detached; object registry had 1 entries
I	08-30 13:59:54.593	2559		dalvikvm	JNI: AttachCurrentThread (from ???.???)
I	08-30 13:59:54.593	2559		AndroidRuntime	NOTE: attach of thread 'Binder Thread #3' failed
D	08-30 13:59:55.002	2571		AndroidRuntime	>>>>>> AndroidRuntime START com.android.internal.os.RuntimeIn
D	08-30 13:59:55.002	2571		AndroidRuntime	CheckJNI is ON
D	08-30 13:59:55.642	2571		AndroidRuntime	Calling main entry com.android.commands.am.Am
I	08-30 13:59:55.682	61	system_process	ActivityManager	Starting: Intent { act=android.intent.action.MAIN cat=[androi
D	08-30 13:59:55.732	2571		AndroidRuntime	Shutting down VM
D	08-30 13:59:55.742	2571		dalvikvm	GC_CONCURRENT freed 103K, 69% free 319K/1024K, external 0K/0K
D	08-30 13:59:55.762	2571		dalvikvm	Debugger has detached; object registry had 1 entries
I	08-30 13:59:55.792	2571		dalvikvm	JNI: AttachCurrentThread (from ???.???)
I	08-30 13:59:55.792	2571		AndroidRuntime	NOTE: attach of thread 'Binder Thread #3' failed
I	08-30 13:59:55.862	2503	com.supermario.helloandroid	HelloAndroid	onCreate!
I	08-30 13:59:56.842	61	system_process	ActivityManager	Displayed com.supermario.helloandroid/.HelloAndroidActivity:
D	08-30 13:59:57.152	142	com.android.launcher	dalvikvm	GC_EXPLICIT freed 141K, 49% free 3079K/5959K, external 4979K/
D	08-30 14:00:05.111	142	com.android.launcher	dalvikvm	GC_EXPLICIT freed 47K, 50% free 3032K/5959K, external 4928K/5

图 1.19　onCreate log 信息

还有一个在调试过程中使用频率很高的工具是 File Explore，如图 1.20 所示，通过这个工具我们可以方便地查看模拟器的文件系统。我们可以查看文件及文件夹的名称、文件的大小、文件最后修改的日期、文件最后修改的时间、文件的读写权限等。我们也可以通过

右上角的按钮分别实现新建/删除文件、从模拟器中复制文件、往模拟器中复制文件等。

Threads | Heap | Allocation Tracker | Network Statistics | File Explorer | Console

Name	Size	Date	Time	Permissions	Info
data		2012-05-16	07:31	drwxrwx--x	
anr		2012-08-22	04:17	drwxrwxr-x	
app		2012-08-30	13:59	drwxrwx--x	
app-private		2012-05-02	01:03	drwxrwx--x	
backup		2012-08-30	12:29	drwx------	
dalvik-cache		2012-08-30	13:59	drwxrwx--x	
data		2012-08-30	13:59	drwxrwx--x	
dontpanic		2012-05-02	01:03	drwxr-x---	
local		2012-05-02	01:03	drwxrwx--x	
lost+found		2012-05-02	01:03	drwxrwx---	
misc		2012-05-02	01:03	drwxrwx--t	
property		2012-07-27	04:58	drwx------	
secure		2012-05-02	01:04	drwx------	
system		2012-08-30	13:59	drwxrwxr-x	
tombstones		2012-05-17	06:01	drwxr-xr-x	
mnt		2012-08-30	12:28	drwxrwxr-x	
system		2011-02-03	22:51	drwxr-xr-x	

图 1.20　File Explore

1.5　其他工具的使用

除了使用 Eclipse 的 DDMS 进行调试外，我们还可以使用 ADB 工具来进行调试。ADB 的全称为 Android Debug Bridge，借助这个工具，我们可以管理设备或手机模拟器的状态，还可以进行以下的操作：

- 快速更新设备或手机模拟器中的代码，如应用或 Android 系统升级；
- 在设备上运行 Shell 命令；
- 管理设备或手机模拟器上的预定端口；
- 在设备或手机模拟器上复制或粘贴文件。

ADB 的工作方式比较特，殊采用监听 Socket TCP 5554 等端口的方式让 IDE 和 Qemu 通信，默认情况下 ADB 会监听相关的网络端口，所以当我们运行 Eclipse 时 ADB 进程就会自动运行，在 Eclipse 中通过 DDMS 来调试 Android 程序，也可以通过手动方式调用。

ADB 工具的功能很多，以下就几个常用的功能略作说明。

（1）查看当前 ADB 的版本

在终端输入 adb version，可以查看到当前 ADB 的版本，如图 1.21 所示。

```
E:\android-sdk-windows\platform-tools>adb version
Android Debug Bridge version 1.0.29
```

图 1.21　查看 ADB 版本

（2）查看当前运行的模拟器

在终端输入 adb devices，可以看到当前运行的模拟器，如图 1.22 所示。

```
E:\android-sdk-windows\platform-tools>adb devices
List of devices attached
emulator-5554   device
```

图 1.22　查看当前运行的模拟器

（3）安装软件

命令格式为 adb install <apk 文件路径>，这个命令将指定的 apk 文件安装到设备上。

（4）卸载软件

命令格式为 adb uninstall <软件名>或者 adb uninstall -k <软件名>，-k 参数，表示卸载软件但是保留配置和缓存文件。

（5）进入设备或模拟器的 shell

命令格式为 adb shell，通过这个命令，就可以进入设备或模拟器的 shell 环境中。在这个 Linux Shell 中，你可以执行各种 Linux 的命令，另外如果只想执行一条 shell 命令，可以采用以下的方式：

```
adb shell [command]
```

例如，adb shell ls 会显示出当前目录下的文件信息。

（6）发布端口

可以设置任意的端口号，作为主机向模拟器或设备的请求端口。如：

```
adb forward tcp:5555 tcp:8000
```

（7）从电脑上发送文件到设备

命令格式为 adb push <本地路径> <远程路径>，用 push 命令可以把本机电脑上的文件或者文件夹复制到设备（手机）。

（8）从设备上下载文件到电脑

命令格式为 adb pull <远程路径> <本地路径>，用 pull 命令可以把设备（手机）上的文件或者文件夹复制到本机电脑。

（9）记录无线通讯日志

一般来说，无线通讯的日志非常多，在运行时没必要去记录，但我们还是可以通过 adb shell 和 logcat -b radio 命令来设置记录。

1.6　本 章 小 结

本章主要介绍了 Android 开发环境的搭建，以及如何利用 Eclipse 自带的 DDMS 工具及 SDK 中的 ADB 工具调试程序。本章我们学会了如何去编写一个最简单的 Android 应用程序——HelloAndroid，以及如何在模拟器中查看运行的结果，在 logcat 工具中查看自己写的 log。

第 2 章　Android 程序开发基础

上一章我们讲述了 Android 平台的框架和特性以及 Android 开发环境的搭建，这部分虽然繁琐，但却是学习 Android 开发必须首先了解的。接下来这章我们要讲解 Android 程序开发必备的一些基础知识。

2.1　Android 项目结构分析

我们通过 Eclipse 创建 Android 工程的时候，Eclipse 会自动为我们生成一个目录结构，这个目录结构是我们今后开发所有应用程序的“原型”，因此我们有必要深入了解一下这个目录结构的各个部分。

上一章我们建立了一个 HelloAndroid 的项目，现在我们把它全部展开，如图 2.1 所示。

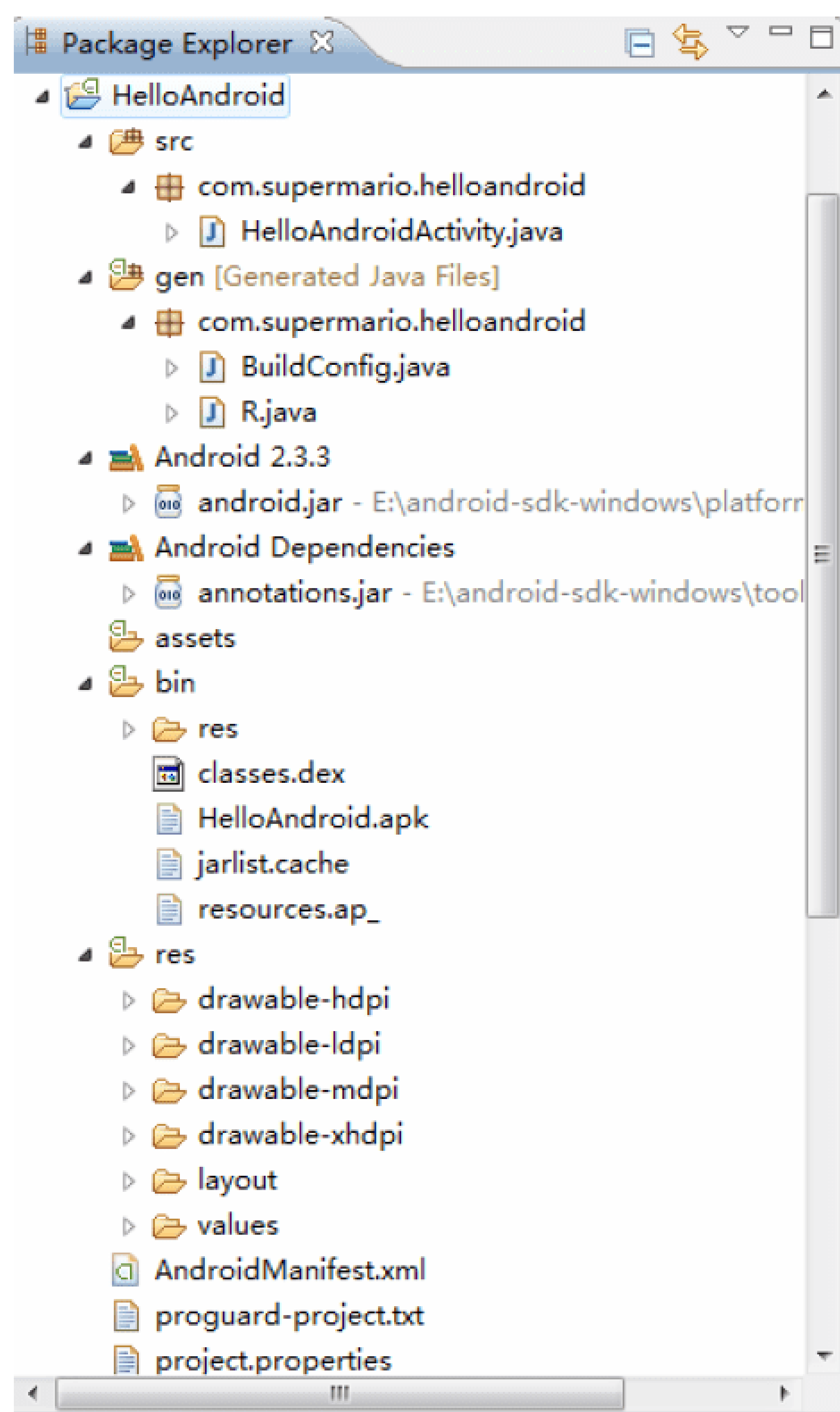


图 2.1　HelloAndroid 项目结构

（1）src：和普通的 Java 项目一样，这是项目所有的包和源文件存放的目录。

（2）res：资源（Resource）目录。

在这个目录中我们可以存放应用使用到的各种资源，如 xml 界面文件、图标或常量。

- res/drawable——专门存放图标文件。
- res/layout——专门存放 xml 界面文件，xml 界面文件和 HTML 文件一样、主要用于用户界面显示。
- res/values——专门存放应用使用到的各种常量，作用和 struts 中的国际化资源文件一样。

（3）gen：gen 目录中存放所有由 Android 开发工具自动生成的文件。目录中最重要的就是 R.java 文件，这个文件是由 Android 开发工具自动产生的。Android 开发工具会自动根据你放入 res 目录的 xml 界面文件、图标与常量，同步更新修改 R.java 文件。正因为 R.java 文件是由开发工具自动生成的，所以我们应避免手工修改 R.java。R.java 在应用中起到了字典的作用，它包含了界面、图标、常量等各种资源的 id，通过 R.java，应用可以很方便地找到对应资源。另外编绎器也会检查 R.java 列表中的资源是否被使用到，没有被使用到的资源不会编译进软件中，这样可以减少应用在手机中占用的空间。

HelloAndroid 的 R.java 文件代码如下所示，里面包含类 drawable、layout、string，分别对应了 res 目录中相应的 drawable、layout、values 中的文件。

```
01  /* AUTO-GENERATED FILE.  DO NOT MODIFY.
02   *
03   * This class was automatically generated by the
04   * aapt tool from the resource data it found.  It
05   * should not be modified by hand.
06   */
07
08  package com.supermario.helloandroid;
09  <!-- Eclipse 自动生成-->
10  public final class R {
11      public static final class attr {
12      }
13      public static final class drawable {
14          public static final int ic_launcher=0x7f020000;
15      }
16      public static final class layout {
17          public static final int main=0x7f030000;
18      }
19      public static final class string {
20          public static final int app name=0x7f040001;
21          public static final int hello=0x7f040000;
22      }
23  }
```

（4）AndroidManifest.xml：功能清单文件。

如以下代码所示，这个文件列出了应用程序所提供的所有功能。每当你添加一个新的 Activity 时，就需要在这个文件中作相应的配置，否则应用程序就无法识别和使用这个 Activity。在这里，你也可以指定应用程序需要用到的服务、接收器等，以及它们对应的 <Intent-filter>，用于通过 intent 的方式打开指定服务或接收器。此外，程序需要的权限也需要在这里设置。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
03     package="com.supermario.helloandroid"
04     android:versionCode="1"
05     android:versionName="1.0" >
06  <!--使用的 SDK 最低版本-->
07     <uses-sdk android:minSdkVersion="10" />
08
09     <application
10         android:icon="@drawable/ic_launcher"
11         android:label="@string/app_name" >
12  <!--主体应用程序-->
13         <activity
14             android:name=".HelloAndroidActivity"
15             android:label="@string/app name" >
16             <intent-filter>
17                 <action android:name="android.intent.action.MAIN" />
18  <!--设置为默认类别-->
19                 <category android:name="android.intent.category.
                   LAUNCHER" />
20             </intent-filter>
21         </activity>
22     </application>
23  </manifest>
24
```

表 2.1 详细描述了 AndroidManifest 部分标签及元素的作用。

表 2.1　AndroidManifest.xml元素功能介绍

manifest	根节点，描述了 package 中所有的内容
xmlns:android	包含命名空间的声明。xmlns:android=http://schemas.android.com/apk/res/android，使得 Android 中各种标准属性能在文件中使用，提供了大部分元素中的数据
application	包含 package 中 application 级别组件声明的根节点。此元素也可包含 application 的一些全局和默认的属性，如标签、icon、主题、必要的权限，等等。一个 manifest 能包含零个或一个此元素（不能大于一个）
android:icon	应用程序图标
android:label	应用程序名字
Activity	用来与用户交互的主要工具。Activity 是用户打开一个应用程序的初始页面，大部分被使用到的其他页面也由不同的 activity 所实现，并声明在另外的 activity 标记中。注意，每一个 activity 必须有一个<activity>标记对应，无论它给外部使用或是只用于自己的 package 中。如果一个 activity 没有对应的标记，你将不能运行它。另外，为了支持运行时查找 Activity，可包含一个或多个<intent-filter>元素来描述 activity 所支持的操作
android:name	应用程序默认启动的 activity
intent-filter	声明了指定的一组组件支持的 Intent 值，从而形成了 IntentFilter。除了能在此元素下指定不同类型的值，属性也能放在这里来描述一个操作所需的唯一的标签、icon 和其他信息
action	组件支持的 Intent action
category	组件支持的 Intent Category。这里指定了应用程序默认启动的 activity
uses-sdk	该应用程序所使用的 SDK 版本信息

（5）default properties：系统默认信息，一般不需要修改此文件。

（6）assets：Android 系统为每个新设计的程序提供了 assets 目录，这个目录保存的文件可以打包在程序里。它与 res 目录不同点在于，Android 不为 assets 下的文件生成 id，如

果要使用 assets 下的文件，需要指定文件的路径和文件名。

（7）bin：存放每次编译生成的文件，编译生成的 HelloAndroid.apk 就存放在这个目录。

2.2　Android 四大组件

Android 开发四大组件分别是：

- ❑ 活动（Activity）：用于表现功能。
- ❑ 服务（Service）：后台运行服务，不提供界面呈现。
- ❑ 广播接收器（BroadcastReceiver）：用于接收广播。
- ❑ 内容提供商（ContentProvider）：支持在多个应用中存储和读取数据，相当于数据库。

2.2.1　Activity

在 Android 中，Activity 是所有程序的根本，所有程序的流程都运行在 Activity 之中，Activity 可以算是开发者遇到的最频繁，也是 Android 当中最基本的模块之一。在 Android 的程序当中，Activity 一般代表手机屏幕的一屏。如果把手机比作一个浏览器，那么 Activity 就相当于一个网页。在 Activity 当中可以添加一些 Button、Check box 等控件。可以看到 Activity 概念和网页的概念相当类似。

一般一个 Android 应用是由多个 Activity 组成的。这多个 Activity 之间可以进行相互跳转，例如，按下一个 Button 按钮后，可能会跳转到其他的 Activity。和网页跳转稍微有些不一样的是，Activity 之间的跳转可能有返回值。例如，从 Activity A 跳转到 Activity B，那么当 Activity B 运行结束的时候，有可能会给 Activity A 一个返回值。这样做在很多时候是相当方便的。

当打开一个新的屏幕时，之前一个屏幕会被置为暂停状态，并且压入历史堆栈中。用户可以通过回退操作返回到以前打开过的屏幕。我们可以选择性地移除一些没有必要保留的屏幕，因为 Android 会把每个应用从开始到当前的每个屏幕都保存在堆栈中。

Android 用 Intent 这个特殊类实现在 Activity 与 Activity 之间的切换。Intent 类用于描述应用的功能。在 Intent 的描述结构中，有两个最重要的部分：动作和动作对应的数据。典型的动作类型有：MAIN、VIEW、PICK、EDIT 等，而动作对应的数据则以 URI 的形式表示。例如，要查看一个人的联系方式，需要创建一个动作类型为 VIEW 的 Intent，以及一个表示这个人的 URI。

通过解析各种 Intent，从一个屏幕导航到另一个屏幕是很简单的。当向前导航时，Activity 将会调用 StartActivity(Intent my Intent)方法。然后，系统会在所有已安装的应用程序中定义的 IntentFilter 中查找，找到最匹配 myIntent 的 Intent 对应的 Activity。新的 Activity 接收到 MyIntent 的通知后，开始运行。当 StartActivity 方法被调用时，将触发解析 MyIntent 的动作，该机制提供了两个关键好处：

- ❑ Activity 能够重复利用从其他组件中以 Intent 的形式产生的请求。
- ❑ Activity 可以在任何时候被具有相同 IntentFilter 的新的 Activity 取代。

下面我们举例来说明两个 Activity 之间的切换。在 Eclipse 中新建一个项目 IntentTest，新建两个 Activity：IntentTest Activity 和 AnotherActivity。IntentActivity.java 代码如下所示，定义了一个按钮并为这个按钮绑定监听事件，当用户单击这个按钮的时候，将会切换到 AnotherActivity，同时关闭当前的 Activity。

```
01  package com.supermario.intenttest;
02  import android.app.Activity;
03  import android.content.Context;
04  import android.content.Intent;
05  import android.os.Bundle;
06  import android.view.View;
07  import android.view.View.OnClickListener;
08  import android.widget.Button;
09  public class IntentTestActivity extends Activity {
10      /** Called when the activity is first created. */
11      @Override
12      public void onCreate(Bundle savedInstanceState) {
13          super.onCreate(savedInstanceState);
14          //设置显示 main 的布局
15          setContentView(R.layout.main);
16          //取得布局 main 中的 button1 按钮
17          Button btn=(Button)findViewById(R.id.button1);
18          //取得当前程序的上下文
19          final Context context=this;
20          //设置 btn 的按键监听器
21          btn.setOnClickListener(new OnClickListener(){
22              @Override
23              public void onClick(View v) {
24                  // TODO Auto-generated method stub
25                  //新建一个 Intent 对象
26                  Intent it=new Intent();
27                  //指定 it 要启动的类
28                  it.setClass(context, AnotherActivity.class);
29                  //启动 it 指定的类
30                  startActivity(it);
31                  //关闭当前的 Activity
32                  finish();
33              }
34          });
35      }
36  }
```

AnotherActivity.java 的代码如下所示，同样定义了一个按钮并为该按钮绑定监听器，当用户单击这个按钮的时候，就会启动 IntentTestActivity，并关闭当前的 Activity。

```
01  package com.supermario.intenttest;
02  import android.app.Activity;
03  import android.content.Context;
04  import android.content.Intent;
05  import android.os.Bundle;
06  import android.view.View;
07  import android.view.View.OnClickListener;
08  import android.widget.Button;
09  public class AnotherActivity extends Activity {
10      /** Called when the activity is first created. */
11      @Override
12      public void onCreate(Bundle savedInstanceState) {
```

```
13          super.onCreate(savedInstanceState);
14          //设置当前的布局为 main2
15          setContentView(R.layout.main2);
16          //取得布局 main2 中的 button1 按钮
17          Button btn=(Button)findViewById(R.id.button1);
18          //取得当前程序的上下文
19          final Context context=this;
20          //设置 btn 的按键监听器
21          btn.setOnClickListener(new OnClickListener(){
22              @Override
23              public void onClick(View v) {
24                  // TODO Auto-generated method stub
25                  //新建一个 Intent 对象
26                  Intent it=new Intent();
27                  //指定 it 要启动的类
28                  it.setClass(context, IntentTestActivity.class);
29                  //启动 it 指定的类
30                  startActivity(it);
31                  //关闭当前的 Activity
32                  finish();
33              }
34          });
35      }
36  }
```

布局文件 main.xml 代码如下，定义了一个 TextView 和 Button。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout_width="fill_parent"
04      android:layout_height="fill_parent"
05      android:orientation="vertical" >
06      <!--用于显示文字-->
07      <TextView
08          android:layout width="fill parent"
09          android:layout height="wrap content"
10          android:text="第一个 Activity! " />
11      <!--切换按钮-->
12      <Button
13          android:id="@+id/button1"
14          android:layout width="wrap content"
15          android:layout height="wrap content"
16          android:text="切换到第二个 Activity" />
17  </LinearLayout>
```

布局文件 main2.xml 代码如下，同样是定义了一个 TextView 和 Button。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout_width="fill_parent"
04      android:layout_height="fill_parent"
05      android:orientation="vertical" >
06      <!--用于显示文字-->
07      <TextView
08          android:layout width="fill parent"
09          android:layout height="wrap content"
10          android:text="第二个 Activity! " />
```

```
11      <!--用于显示按钮-->
12     <Button
13         android:id="@+id/button1"
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:text="切换到第一个 Activity" />
17   </LinearLayout>
```

在 Androidmanifest.xml 中声明使用 IntentTestActivity 和 AnotherActivity，如下所示：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03      package="com.supermario.intenttest"
04      android:versionCode="1"
05      android:versionName="1.0" >
06      <!--使用的最低 SDK 版本-->
07      <uses-sdk android:minSdkVersion="10" />
08      <application
09          android:icon="@drawable/ic_launcher"
10          android:label="@string/app_name" >
11          <!--主体 Activity-->
12          <activity
13              android:name=".IntentTestActivity"
14              android:label="@string/app name" >
15              <intent-filter>
16                  <action android:name="android.intent.action.MAIN" />
17                  <category android:name="android.intent.category.
                    LAUNCHER" />
18              </intent-filter>
19          </activity>
20          <!--声明另一个 Activity-->
21          <activity android:name=".AnotherActivity">
22              <intent-filter>
23                  <action android:name="android.guo.action.anotherac-
                    tivity" />
24              </intent-filter>
25          </activity>
26      </application>
27  </manifest>
```

最后我们运行该程序，得到如图 2.2 所示的效果，单击“切换到第二个 Activity”，可以切换到第二个 AnotherActivity，如图 2.3 所示。再单击“切换到第一个 Activity”可以切换回 IntentTestActivity。



图 2.2　IntentTestActivity



图 2.3　AnotherActivity

2.2.2　Service

Service 是 Android 系统中的一种组件，它跟 Activity 的级别差不多，但是它不能自己运行，只能后台运行，并且可以和其他组件进行交互。Service 是没有界面的长生命周期的代码。Service 是一种程序，它可以运行很长时间，但是它却没有用户界面。这么说有点枯燥，来看个例子，打开一个音乐播放器的程序，这个时候若想上网了，那么，我们打开 Android 浏览器，这个时候虽然我们已经进入了浏览器程序，但是，歌曲播放并没有停止，而是在后台继续一首接着一首地播放。其实这个播放就是由播放音乐的 Service 进行控制的。

当然这个播放音乐的 Service 也可以停止，例如，当播放列表里边的歌曲都结束，或者用户按下了停止音乐播放的快捷键等。Service 可以在很多场合的应用中使用，比如播放多媒体的时候用户启动了其他 Activity，这个时候程序要在后台继续播放，比如检测 SD 卡上文件的变化，再或者在后台记录用户地理信息位置的改变等等。总之服务嘛，总是藏在后头的。

开启 Service 有两种方式。

（1）Context.startService()：Service 会经历 onCreate→onStart（如果 Service 还没有运行，则 Android 先调用 onCreate()然后调用 onStart()；如果 Service 已经运行，则只调用 onStart()，所以一个 Service 的 onStart 方法可能会重复调用多次）；stopService 的时候直接 onDestroy，如果是调用者自己直接退出而没有调用 stopService 的话，Service 会一直在后台运行。该 Service 的调用者再启动起来后可以通过 stopService 关闭 Service。注意，多次调用 Context.startService()不会嵌套（即使会有相应的 onStart()方法被调用），所以无论同一个服务被启动了多少次，一旦调用 Context.stopService()或者 stopSelf()，它都会被停止。补充说明：传递给 startService()的 Intent 对象会传递给 onStart()方法。调用顺序为：onCreate→onStart（可多次调用）→onDestroy。

（2）Context.bindService()：Service 会经历 onCreate()→onBind()，onBind 将返回给客户端一个 IBind 接口实例，IBind 允许客户端回调服务的方法，比如得到 Service 运行的状态或其他操作。这个时候调用者（Context，如 Activity）会和 Service 绑定在一起，Context 退出了，Srevice 就会调用 onUnbind→onDestroy 相应退出。所谓绑定在一起就共存亡了。

下面我们以一个小例子来学习一下 Service 的用法。在这个例子中，我们在一个 Activity 中设置两个按钮，一个按钮用来启动 Service，一个按钮用来停止 Service。在 Service 中使用相应的函数实现音乐的播放和停止，项目的目录结构如图 2.4 所示。

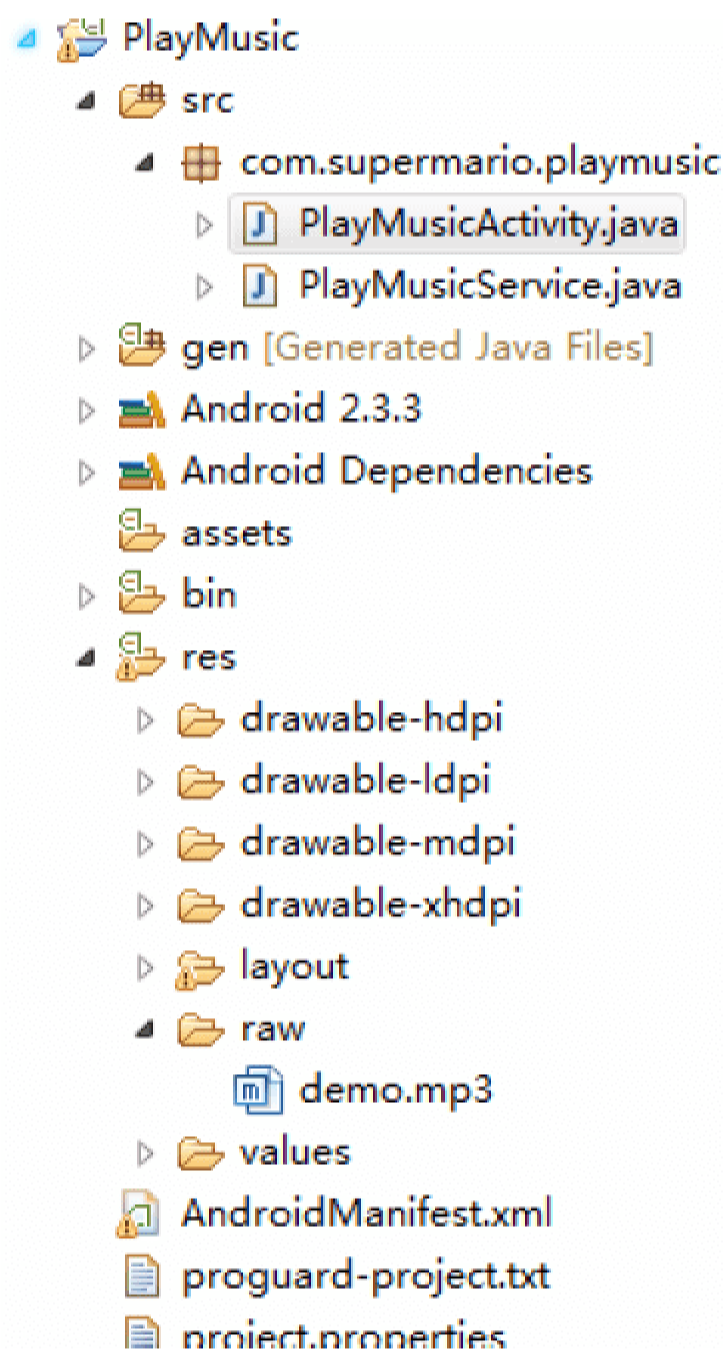


图 2.4　PlayMusic 目录结构

如下是 PlayMusicActivity.java 的代码，在代码中定义了两个按钮，并分别为这两个按钮绑定了监听器，在这两个监

听器中分别打开和关闭指定的 Service。

```
01  package com.supermario.playmusic;
02  import android.app.Activity;
03  import android.content.Intent;
04  import android.os.Bundle;
05  import android.view.View;
06  import android.view.View.OnClickListener;
07  import android.widget.Button;
08  public class PlayMusicActivity extends Activity {
09      /** Called when the activity is first created. */
10      @Override
11      public void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.main);
14          //开始按钮
15          Button start=(Button)findViewById(R.id.button1);
16          //停止按钮
17          Button stop=(Button)findViewById(R.id.button2);
18          //用于启动和停止 Service 的 Intent
19          final Intent it=new Intent("android.guo.service.playmusic");
20          //为"开始"按钮绑定监听器
21          start.setOnClickListener(new OnClickListener(){
22              @Override
23              public void onClick(View v) {
24                  // TODO Auto-generated method stub
25                  //启动服务
26                  startService(it);
27              }
28          });
29          //为"停止"按钮绑定监听器
30          stop.setOnClickListener(new OnClickListener(){
31              @Override
32              public void onClick(View v) {
33                  // TODO Auto-generated method stub
34                  //停止服务
35                  stopService(it);
36              }
37          });
38      }
39  }
```

如下是 PlayMusicService.java 的代码，在服务启动的时候将会执行 onStart()，在关闭的时候将会执行 onDestroy()，因此，我们在 onStart()函数中开始播放音乐，在 onDestroy()中停止音乐。程序中用到的音乐是我们预先放到 res/raw/目录下的一个 demo.mp3 文件。

```
01  package com.supermario.playmusic;
02  import android.app.Service;
03  import android.content.Intent;
04  import android.media.MediaPlayer;
05  import android.os.IBinder;
06  public class PlayMusicService extends Service{
07      //定义音乐播放器
08      private MediaPlayer mMediaPlayer;
09      @Override
10      public IBinder onBind(Intent arg0) {
11          // TODO Auto-generated method stub
12          return null;
13      }
```

```
14      @Override
15      public void onStart(Intent intent,int startId)
16      {
17          super.onStart(intent, startId);
18          //装载音乐
19          mMediaPlayer=MediaPlayer.create(this, R.raw.demo);
20          //开始播放音乐
21          mMediaPlayer.start();
22      }
23      @Override
24      public void onDestroy()
25      {
26          super.onDestroy();
27          //停止播放音乐
28          mMediaPlayer.stop();
29      }
30  }
```

在布局文件中，定义两个按钮，一个 ID 为 start，另一个 ID 为 stop，分别用于播放音乐和停止音乐，如下所示。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout width="fill parent"
04      android:layout height="fill parent"
05      android:orientation="vertical" >
06      <TextView
07          android:layout width="fill parent"
08          android:layout height="wrap content"
09          android:text="音乐播放器--回到拉萨" />
10      <!-- 播放音乐 -->
11      <Button
12          android:id="@+id/button1"
13          android:layout width="wrap content"
14          android:layout height="wrap content"
15          android:text="播放" />
16      <!-- 停止音乐 -->
17      <Button
18          android:id="@+id/button2"
19          android:layout width="wrap content"
20          android:layout height="wrap content"
21          android:text="停止" />
22  </LinearLayout>
```

在 Androidmanifest.xml 中增加对这个 Service 的定义，此处的<intent-filter>中的 action 名字可以任意。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03      package="com.supermario.playmusic"
04      android:versionCode="1"
05      android:versionName="1.0" >
06      <!--最低 SDK 版本-->
07      <uses-sdk android:minSdkVersion="10" />
08      <application
09          android:icon="@drawable/ic_launcher"
10          android:label="@string/app_name" >
11          <!--主体 Activity-->
```

```
12          <activity
13              android:name=".PlayMusicActivity"
14              android:label="@string/app name" >
15              <intent-filter>
16                  <action android:name="android.intent.action.MAIN" />
17
18                  <category android:name="android.intent.category.
                    LAUNCHER" />
19              </intent-filter>
20          </activity>
21          <!--声明服务-->
22          <service android:name=".PlayMusicService">
23              <intent-filter>
24                  <action android:name="android.guo.service.playmusic"/>
25              </intent-filter>
26          </service>
27      </application>
28  </manifest>
```

最后我们运行该程序，单击“播放”按钮，可以听到音乐，如图 2.5 所示。这时候就算我们退出了这个程序，音乐也不会停止，表明播放音乐这个动作一直在后台执行着。



图 2.5　PlayMusic

2.2.3　Broadcast

在 Android 中，Broadcast 是一种广泛运用的在应用程序之间传输信息的机制。而 BroadcastReceiver 是对发送出来的 Broadcast 进行过滤接收并响应的一类组件。可以使用 BroadcastReceiver 来让应用对一个外部的事件作出响应。这是非常有意思的，例如，当电话呼入这个外部事件到来的时候，可以利用 BroadcastReceiver 进行处理。又如，当下载一个程序成功完成的时候，仍然可以利用 BroadcastReceiver 进行处理。BroadcastReceiver 不能生成 UI，因此用户看不到相应的界面。

BroadcastReceiver 通过 NotificationManager 来通知用户这些事情发生了。BroadcastReceiver 既可以在 AndroidManifest.xml 中注册，也可以在运行时的代码中使用 Context.registerReceiver()进行注册。只要是注册了，当事件来临的时候，即使程序没有启

动，系统也会在需要的时候启动程序。各种应用还可以通过使用 Context.sendBroadcast () 将它们自己的 intent broadcasts 广播给其他应用程序。

注册 BroadcastReceiver 有两种方式。

（1）在 AndroidManifest.xml 进行注册。这种方法有一个特点，即使你的应用程序已经关闭了，但这个 BroadcastReceiver 依然会接受广播出来的对象，也就是说无论这个应用程序是开还是关，都属于活动状态，都可以接收到广播的事件。

（2）在代码中注册广播。

第一种俗称静态注册，第二种俗称动态注册，这两种注册 BroadcastReceiver 的区别是：动态注册较静态注册灵活。实验证明，当静态注册一个 BroadcastReceiver 时，不论应用程序启动与否，都可以接受对应的广播。动态注册的时候，如果不执行 unregisterReceiver() 方法取消注册，跟静态是一样的。但是如果执行该方法，当执行过以后，就不能接受广播了。

下面我们看一下静态绑定广播的一个小例子，项目结构如图 2.6 所示。

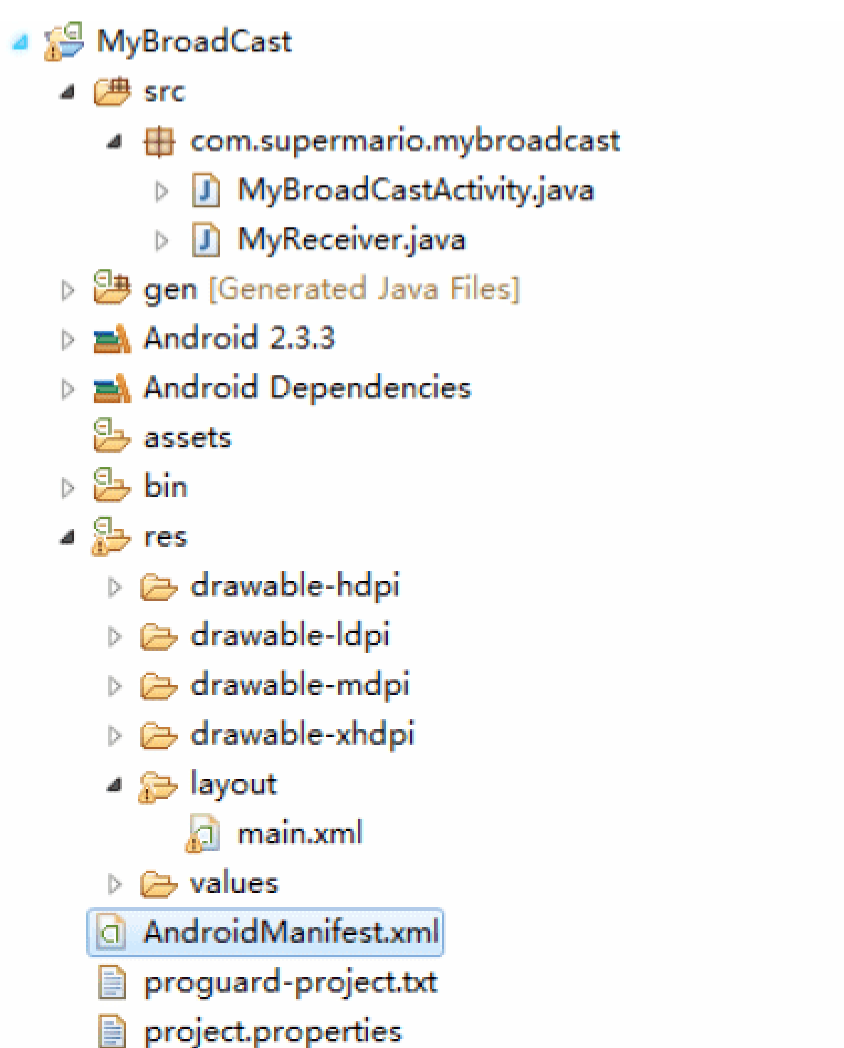


图 2.6　MyBroadCast 目录结构

如下所示，我们在 MyBroadCastActivity 中为按钮设置监听器，当单击这个按钮的时候，将会发送一个广播，这个广播的对象由 Intent 的内容来决定，我们设置这个 Intent 对应的 action 为 com.guo.receiver.myreceiver，这个 action 的名字要和我们在 AndroidManifest.xml 中声明的名字保持一致。

```
01  package com.supermario.mybroadcast;
02  import android.app.Activity;
03  import android.content.Intent;
04  import android.os.Bundle;
05  import android.view.View;
06  import android.view.View.OnClickListener;
07  import android.widget.Button;
08  public class MyBroadCastActivity extends Activity {
09      /** Called when the activity is first created. */
10      @Override
11      public void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
```

```
13          setContentView(R.layout.main);
14          //发送广播按钮
15          Button btn=(Button)findViewById(R.id.button1);
16          //设置发送广播对应的 Intent
17          final Intent intent=new Intent("com.guo.receiver.myreceiver");
18          btn.setOnClickListener(new OnClickListener(){
19              @Override
20              public void onClick(View arg0) {
21                  // TODO Auto-generated method stub
22                  //发送广播
23                  sendBroadcast(intent);
24              }
25          });
26      }
27  }
```

同时我们新建一个广播接收器——MyReceiver，代码如下所示，我们在接收器的onReceive ()函数中使用一个 Toast 来显示一条消息，表明我们收到了相应的广播。

```
01  package com.supermario.mybroadcast;
02  import android.content.BroadcastReceiver;
03  import android.content.Context;
04  import android.content.Intent;
05  import android.widget.Toast;
06  public class MyReceiver extends BroadcastReceiver {
07      @Override
08      public void onReceive(Context arg0, Intent arg1) {
09          // TODO Auto-generated method stub
10          Toast.makeText(arg0, "成功接收广播! ", Toast.LENGTH_SHORT)
            .show();
11      }
12  }
```

在布局文件 main.xml 中，简单增加一个按钮，如下所示。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout_width="fill_parent"
04      android:layout_height="fill_parent"
05      android:orientation="vertical" >
06      <TextView
07          android:layout_width="fill_parent"
08          android:layout_height="wrap_content"
09          android:text="@string/hello" />
10      <!-- 用于发送广播 -->
11      <Button
12          android:id="@+id/button1"
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:text="发送广播" />
16  </LinearLayout>
```

最后，很重要的一点，就是 Androidmanifest.xml 中声明这个 receiver，并将 action 设置为我们前面使用的 com.guo.receiver.myreceiver。这样，我们这个 receiver 就能正确匹配到前面设置的 Intent。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
03      package="com.supermario.mybroadcast"
04      android:versionCode="1"
05      android:versionName="1.0" >
06      <!--声明 SDK 的最低版本-->
07      <uses-sdk android:minSdkVersion="10" />
08      <application
09          android:icon="@drawable/ic_launcher"
10          android:label="@string/app_name" >
11          <!--声明主体 Activity-->
12          <activity
13              android:name=".MyBroadCastActivity"
14              android:label="@string/app name" >
15              <intent-filter>
16                  <action android:name="android.intent.action.MAIN" />
17
18                  <category android:name="android.intent.category.
                    LAUNCHER" />
19              </intent-filter>
20          </activity>
21          <!-- 声明 receiver -->
22          <receiver android:name=".MyReceiver">
23              <intent-filter>
24                  <action android:name="com.guo.receiver.myreceiver" />
25              </intent-filter>
26          </receiver>
27      </application>
28  </manifest>
```

我们在模拟器中运行该程序，单击“发送广播”，可以看到如图 2.7 所示的结果。

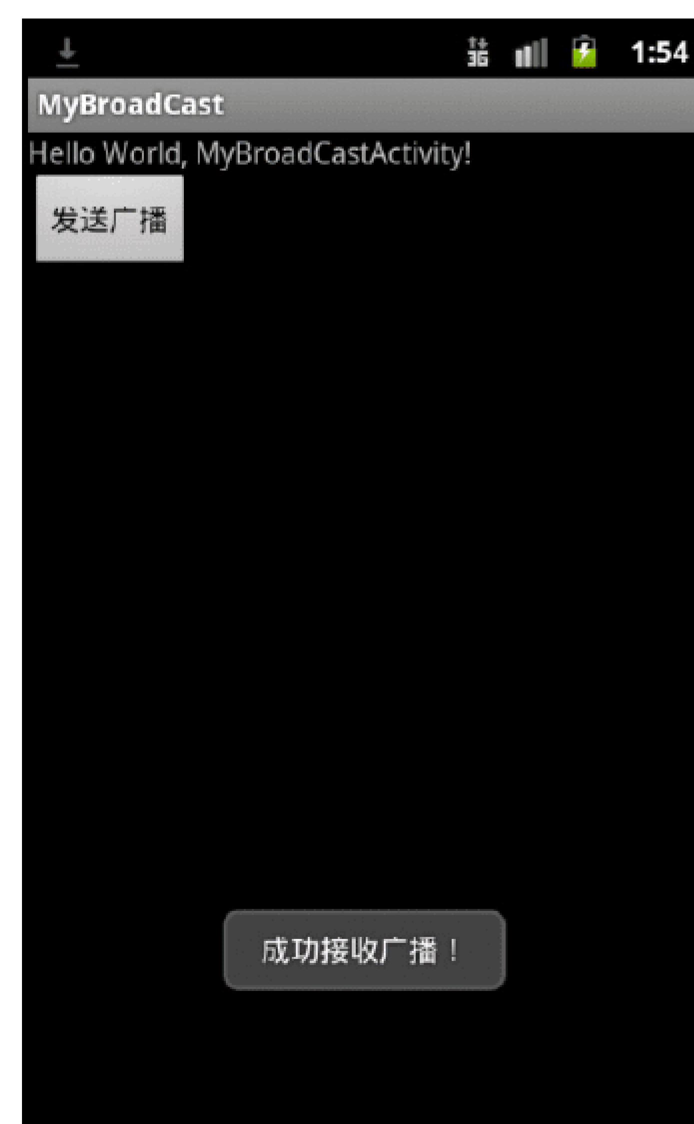


图 2.7　MyBroadCast

2.2.4　Content Provider

Content Provider 是 Android 提供的第三方应用数据的访问方案。在 Android 中，对数据的保护是很严密的，除了放在 SD 卡中的数据，一个应用所持有的数据库、文件等内容，都是不允许其他程序直接访问的。Android 当然不会真的把每个应用都做成一座孤岛，它为所有应用都准备了一扇窗，这就是 Content Provider。应用想对外提供的数据，可以通过派生 Content Provider 类，封装成一枚 Content Provider，每个 Content Provider 都用一个 uri 作为独立的标识，形如：content://com.xxxxx。

所有东西看着像 REST 的样子，但实际上，它比 REST 更为灵活。和 REST 类似，uri 也可以有两种类型，一种是带 id 的，另一种是列表的，但实现者不需要按照这个模式来做，给你 id 的 uri 你也可以返回列表类型的数据，只要调用者明白就无妨，不用苛求所谓的 REST。

另外，Content Provider 不和 REST 一样只有 uri 可用，它还可以接受 Projection、Selection、OrderBy 等参数，这样，就可以像数据库那样进行投影、选择和排序。查询到的结果，以 Cursor 的形式进行返回，调用者可以移动 Cursor 来访问各列的数据。

Content Provider 屏蔽了内部数据的存储细节，向外提供了上述统一的接口模型，这样的抽象层次，大大简化了上层应用的编写，也对数据的整合提供了更方便的途径。Content Provider 内部常用数据库来实现，Android 提供了强大的 Sqlite 支持，但很多时候，你也可以封装文件或其他混合的数据。

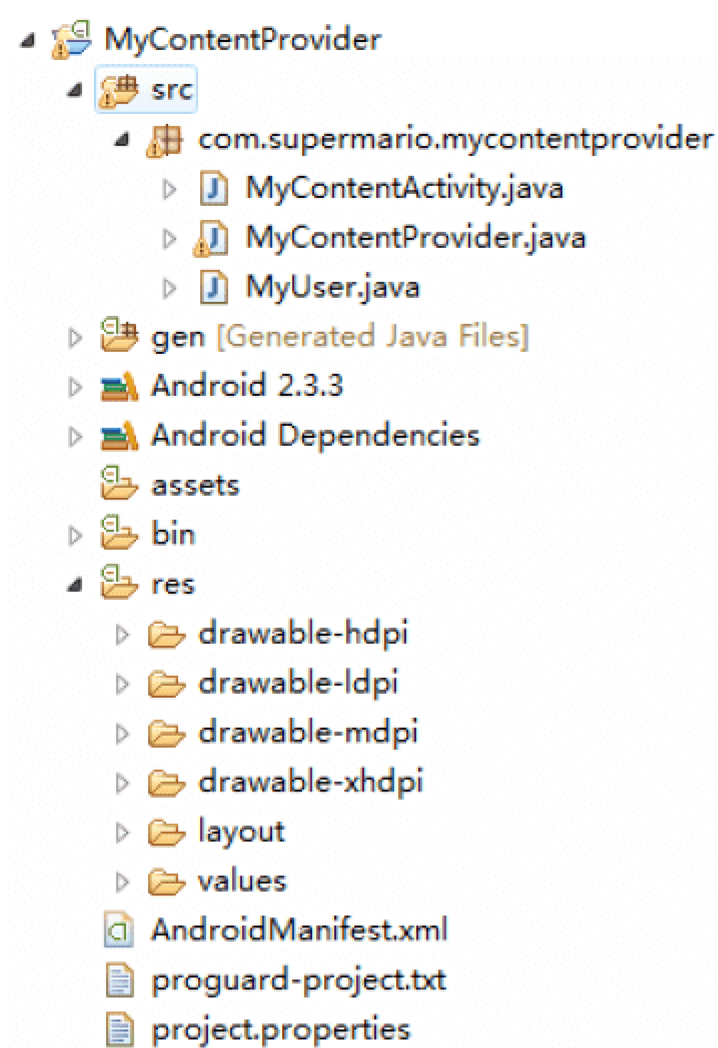


图 2.8　MyContentProvider 目录结构

在 Android 中，ContentResolver 是用来发起 Content Provider 的定位和访问的。不过它仅提供了同步访问的 Content Provider 的接口。但通常，Content Provider 需要访问的可能是数据库等大数据源，效率上不足够快，会导致调用线程的拥塞。因此 Android 提供了一个 AsyncQueryHandler，帮助进行异步访问 Content Provider。

在各大组件中，Service 和 Content Provider 都是需要持续访问的。Service 如果是一个耗时的场景，往往会提供异步访问的接口，而 Content Provider 不论效率如何，都提供的是约定的同步访问接口。

下面以两个实例——一个是 ContentProvider 所在的应用，一个是使用 ContentProvider 的应用，来说明如何使用 ContentProvider。

我们先新建一个工程 MyContentProvider，在这个工程中我们要实现 ContentProvider，项目的目录结构如图 2.8 所示。

在 MyUser.java 中，MyUser 这个类主要用来存放一些常量，比如这个 ContentProvier 的 Uri 以及使用的数据库表的列，如下所示：

```
01  package com.supermario.mycontentprovider;
02  import android.net.Uri;
03  import android.provider.BaseColumns;
04  public class MyUser {
05      public static final String AUTHORITY = "com.supermario.MyContent
        Provider";
06      // BaseColumn 类中已经包含了_id 字段
07      public static final class User implements BaseColumns {
08          // 定义 Uri
09          public static final Uri CONTENT_URI = Uri.parse("content://"
10                  + AUTHORITY);
11          // 定义数据表列
12          public static final String USER_NAME = "USER_NAME";
13      }
14  }
```

接下来我们在 MyContentProvider.java 中实现这个 Content Provider，在 MyContent Provider 中我们主要构造了一个继承于 SQLiteOpenHelper 的类 DatabaseHelper，然后在这个 SQLiteOpenHelper 类中实现数据的增、删、改、查，如下所示：

```
001 package com.supermario.mycontentprovider;
002 import android.content.ContentProvider;
003 import android.content.ContentUris;
004 import android.content.ContentValues;
005 import android.content.Context;
006 import android.database.Cursor;
007 import android.database.SQLException;
```

```
008 import android.database.sqlite.SQLiteDatabase;
009 import android.database.sqlite.SQLiteOpenHelper;
010 import android.database.sqlite.SQLiteQueryBuilder;
011 import android.net.Uri;
012 /**
013  * MyContentProvider 继承 ContentProvider 类，实现其 insert,update,delete,
      getType,onCreate 等方法
014  */
015 public class MyContentProvider extends ContentProvider {
016     // 定义一个 SQLiteDatabase 变量
017     private SQLiteDatabase sqlDB;
018     // 定义一个 DatabaseHelper 变量
019     private DatabaseHelper dbHelper;
020     // 数据库名
021     private static final String DATABASE_NAME = "Users.db";
022     // 数据库版本
023     private static final int DATABASE_VERSION = 1;
024     // 表名
025     private static final String TABLE_NAME = "User";
026     /**
027      * 定义一个内部类
028      *
029      * 这个内部类继承 SQLiteOpenHelper 类，重写其方法
030      */
031     public static class DatabaseHelper extends SQLiteOpenHelper {
032         // 构造方法
033         public DatabaseHelper(Context context) {
034             // 父类构造方法
035             super(context, DATABASE_NAME, null, DATABASE_VERSION);
036         }
037         // 当第一次创建数据库的时候调用该方法，可以为数据库增加一些表和初始化一
            些数据
038         @Override
039         public void onCreate(SQLiteDatabase db) {
040             // 在数据库里生成一张表
041             db.execSQL("Create table "
042                     + TABLE_NAME
043                     + "( _id INTEGER PRIMARY KEY AUTOINCREMENT, USER_NAME
                    TEXT);");
044         }
045         // 当更新数据库版本的时候，调用该方法。可以删除、修改表的一些信息
046         @Override
047         public void onUpgrade(SQLiteDatabase db, int oldVersion, int
            newVersion) {
048             db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
049             onCreate(db);
050         }
051     }
052     // 这是一个回调函数，当生成所在类的对象时，这个方法被调用，创建一个数据库
053     @Override
054     public boolean onCreate() {
055         dbHelper = new DatabaseHelper(getContext());
056         //返回创建对象是否成功
057         return (dbHelper == null) ? false : true;
058     }
059     // 查询
060     @Override
061     public Cursor query(Uri uri, String[] projection, String selection,
```

```
062             String[] selectionArgs, String sortOrder) {
063         //新建数据库查询类
064         SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
065         SQLiteDatabase db = dbHelper.getReadableDatabase();
066         qb.setTables(TABLE_NAME);
067         //取得查询结果的游标
068         Cursor c = qb.query(db, projection, selection, null, null, null,
069                 sortOrder);
070         c.setNotificationUri(getContext().getContentResolver(), uri);
071         return c;
072     }
073     // 取得类型
074     @Override
075     public String getType(Uri uri) {
076         return null;
077     }
078     // 插入数据
079     @Override
080     public Uri insert(Uri uri, ContentValues contentvalues) {
081         sqlDB = dbHelper.getWritableDatabase();
082         long rowId = sqlDB.insert(TABLE_NAME, "", contentvalues);
083         if (rowId > 0) {
084             Uri rowUri = ContentUris.appendId(
085                     MyUser.User.CONTENT_URI.buildUpon(), rowId).build();
086             //通知更改
087             getContext().getContentResolver().notifyChange(rowUri,
            null);
088             return rowUri;
089         }
090         throw new SQLException("Failed to insert row into" + uri);
091     }
092     // 删除数据
093     @Override
094     public int delete(Uri uri, String selection, String[] selectionArgs)
{
095         return 0;
096     }
097     // 更新数据
098     @Override
099     public int update(Uri uri, ContentValues values, String selection,
100             String[] selectionArgs) {
101         return 0;
102     }
103 }
```

另外，我们新建一个 MyContentActivity 的界面，代码如下所示，在这个界面中简单地往数据库中插入两条记录 Test 和 Guo 作为示例数据，主要是验证在客户端中是否能正确读取到这些数据。插入数据之后，我们还通过 Toast 的方式将数据库中的所有数据显示一遍，表明我们已经将数据正确存储到了数据库中。

```
01  package com.supermario.mycontentprovider;
02  import android.app.Activity;
03  import android.content.ContentValues;
04  import android.database.Cursor;
05  import android.net.Uri;
06  import android.os.Bundle;
07  import android.widget.Toast;
08  public class MyContentActivity extends Activity {
09
```

```
10      @Override
11      protected void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
13          //插入两条记录
14          insertRecord("Test");
15          insertRecord("Guo");
16          //显示记录
17          displayRecords();
18      }
19      //插入记录
20      private void insertRecord(String userName) {
21          ContentValues values = new ContentValues();
22          values.put(MyUser.User.USER_NAME, userName);
23          getContentResolver().insert(MyUser.User.CONTENT_URI, values);
24      }
25      private void displayRecords() {
26          //构建一个字符串数组用于存放用户的记录
27          String columns[] = new String[] { MyUser.User._ID,
28                  MyUser.User.USER_NAME };
29          //设定 ContentProvider 的 Uri
30          Uri myUri = MyUser.User.CONTENT_URI;
31          Cursor cur = managedQuery(myUri, columns, null, null, null);
32          if (cur.moveToFirst()) {
33              String id = null;
34              String userName = null;
35              do {
36                  id = cur.getString(cur.getColumnIndex(MyUser.User._ID));
37                  userName = cur.getString(cur
38                          .getColumnIndex(MyUser.User.USER_NAME));
39                  //显示数据表中的数据
40                  Toast.makeText(this, id + " " + userName, Toast.
                    LENGTH_LONG)
41                          .show();
42              } while (cur.moveToNext());
43          }
44      }
45  }
```

这里我们没有用到任何的布局文件，因此我们直接来看看 AndroidManifest.xml，如下所示。与其他 AndroidManifest 的不同点主要在于加了声明 provider 的部分，如代码 11、12 行所示。核心的两个属性分别是 android:name 和 android:authorities。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03      package="com.supermario.mycontentprovider"
04      android:versionCode="1"
05      android:versionName="1.0" >
06      <uses-sdk android:minSdkVersion="10" />
07      <application
08          android:icon="@drawable/ic_launcher"
09          android:label="@string/app_name" >
10          <!-- 声明 ContentProvider -->
11          <provider android:name="MyContentProvider"
12              android:authorities="com.supermario.MyContentProvider">
                </provider>
13          <!--声明主体 Activity-->
14          <activity
15              android:name=".MyContentActivity"
16              android:label="@string/app_name" >
```

```
17              <intent-filter>
18                  <action android:name="android.intent.action.MAIN" />
19                  <category android:name="android.intent.category.
                    LAUNCHER" />
20              </intent-filter>
21          </activity>
22      </application>
23  </manifest>
```

运行之后效果如图 2.9 所示，显示了添加的用户 Test 和 Guo。

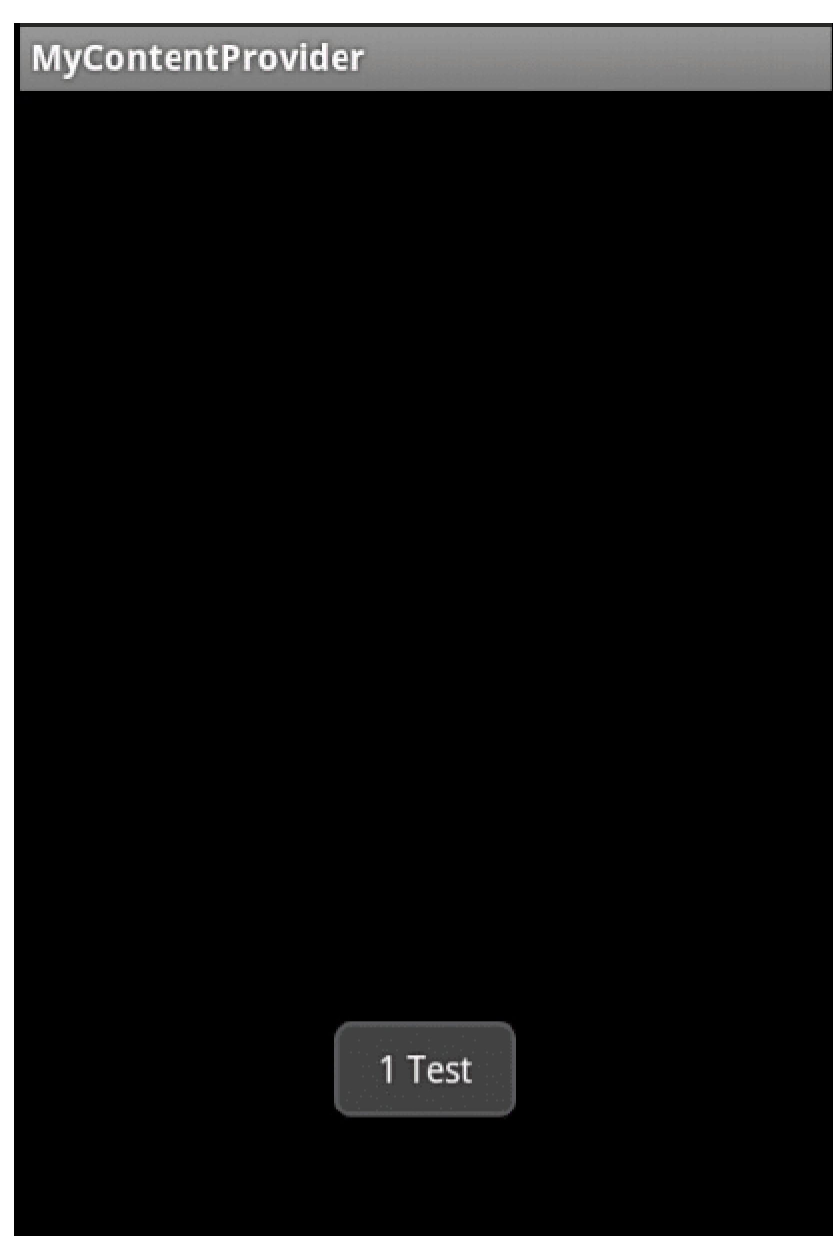


图 2.9　MyContentProvider 运行效果图

接下去我们新建另外一个程序 MyContentClient，在这个程序中使用我们刚才建立的 ContentProvider，对该数据表进行插入和查询操作。

如下所示，是 MyContentClientActivity 的源代码，代码中关键的地方在于要正确设置访问的 Uri 地址，如 16 行所示。这个 Uri 是由“content://”加上我们在 ContentProvider 定义的 authority 组成。然后就是使用 managedQuery 函数得到数据表的所有数据，并保存到一个游标变量中。接着，通过移动这个游标我们可以得到数据表中的所有数据，并将它们保存到一个 StringBuffer 中，最后将它们输出到一个 TextView 控件中。

同时，为了试验插入数据的功能，我们创建了一个 EditText 的文本输入框，在文本框中输入信息，并单击 Button，可以将数据写入到数据库中。数据写入过程也要采用特有的格式：先新建一个 ContentResolver 对象和一个 ContentValue 的对象，再往这个 ContentValue 对象中输入数据，最后通过这个 ContentResolver 对象将数据插入到指定的 ContentProvider 中。当然，这里没有做一个可以实时更新的界面，不过没关系，重新打开程序，就可以在 TextView 控件里看到刚才输入的数据。

```
01  package com.supermario.mycontentclient;
02  import android.app.Activity;
03  import android.content.ContentResolver;
04  import android.content.ContentValues;
05  import android.database.Cursor;
```

```
06 import android.net.Uri;
07 import android.os.Bundle;
08 import android.view.View;
09 import android.widget.Button;
10 import android.widget.EditText;
11 import android.widget.TextView;
12 public class MyContentClientActivity extends Activity {
13     public static final String AUTHORITY = "com.supermario.MyContent
       Provider";
14     private Button insertButton = null;
15     // 访问 ContentProvider 的 Uri
16     Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY);
17     @Override
18     public void onCreate(Bundle savedInstanceState) {
19         super.onCreate(savedInstanceState);
20         setContentView(R.layout.main);
21         TextView show=(TextView)findViewById(R.id.show);
22         StringBuffer sb=new StringBuffer("");
23         // 得到 ContentProvider 对应表的所有数据，以游标格式保存
24         Cursor c = managedQuery(CONTENT_URI,
25                 new String[] { "_id", "USER_NAME" }, null, null, null);
26         // 循环输出 ContentProvider 的数据
27         if (c.moveToFirst()) {
28             String  _id = null;
29             String user_name = null;
30             do {
31                 // 得到_id 列,USER_NAME 列
32                 _id = c.getString(c.getColumnIndex("_id"));
33                 user_name = c.getString(c.getColumnIndex("USER_NAME"));
34
35                 sb.append("_id = " + _id + ", user_name = "
36                         + user_name+"\n");
37             } while (c.moveToNext());
38         }
39         show.setText(sb);
40         // 根据 Id 得到控件对象
41         insertButton = (Button) findViewById(R.id.insert);
42         // 给按钮绑定事件监听器
43         insertButton.setOnClickListener(new View.OnClickListener() {
44             @Override
45             public void onClick(View v) {
46                 // 得到 EditText 输入的数据
47                 String username = ((EditText) findViewById(R.id.user
                   Name))
48                         .getText().toString();
49                 // 生成一个 ContentResolver 对象
50                 ContentResolver cr = getContentResolver();
51                 // 生成一个 ContentValues 对象
52                 ContentValues values = new ContentValues();
53                 // 将 EditText 输入的值，保存到 ContentValues 对象中
54                 values.put("USER_NAME", username);
55                 // 插入数据
56                 cr.insert(CONTENT_URI, values);
57             }
58         });
59     }
60 }
```

按照我们程序的需求，我们在 main.xml 中创建一个 TextView 用于显示数据库信息，

创建一个 EditText 用于输入信息到数据库，最后创建一个输入按钮，用来提交数据。main.xml 的代码如下所示：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout width="fill parent"
04      android:layout height="fill parent"
05      android:orientation="vertical" >
06      <!—用于显示数据库信息 -->
07      <TextView
08          android:id="@+id/show"
09          android:layout_width="fill_parent"
10          android:layout_height="wrap_content" />
11      <!—文本编辑框 ，用于输入信息-->
12      <EditText
13          android:id="@+id/ sername"
14          android:layout width="fill parent"
15          android:layout height="wrap content" />
16      <!—输入按钮-->
17      <Button
18          android:id="@+id/insert"
19          android:layout_width="wrap_content"
20          android:layout_height="wrap_content"
21          android:text="Button" />
22  </LinearLayout>
```

最后我们运行程序，如图 2.10 所示，我们输入“安卓”，单击 Button 进行提交，再次运行程序如图 2.11 所示，刚才输入的数据显示到 TextView 中。

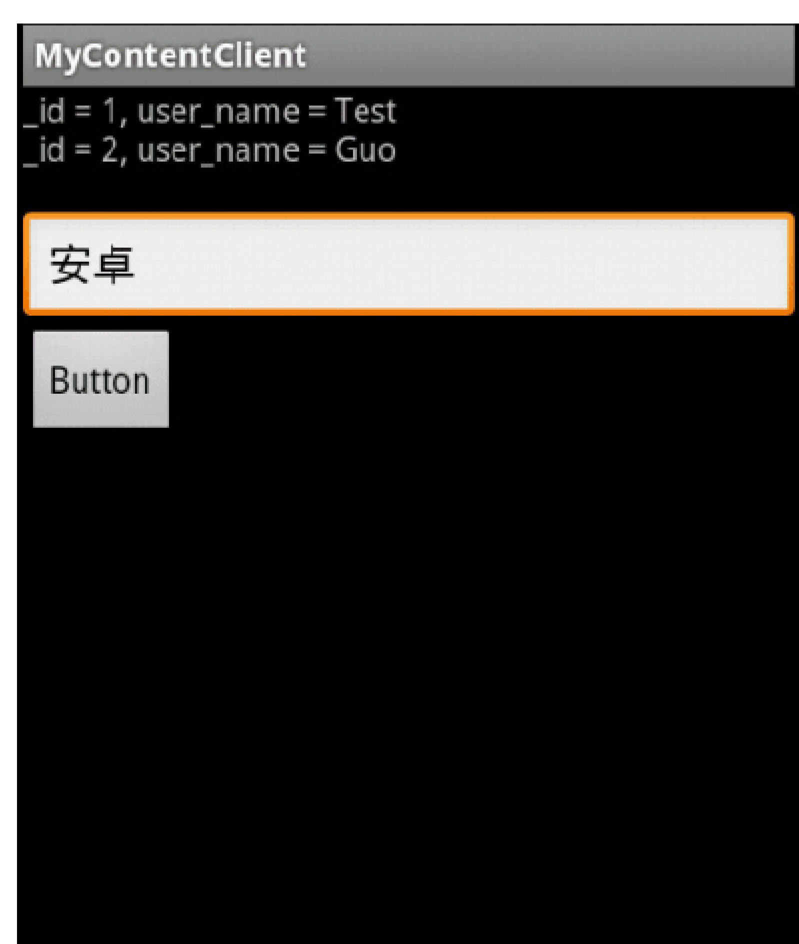


图 2.10　输入数据



图 2.11　查看数据库数据

2.3　Activity 的生命周期

首先我们还是来看一下 Android API 提供的 Activity 生命周期图，如图 2.12 所示。可以看出，一个 Activity 的生命周期会经历 onCreate()→onStart()→onResume()→onPause()→

→onStop()→onDestroy()这几个过程。不过光看图还是有点抽象，下面我们结合一个小例子来熟悉一下 Activity 的生命周期具体的流程。

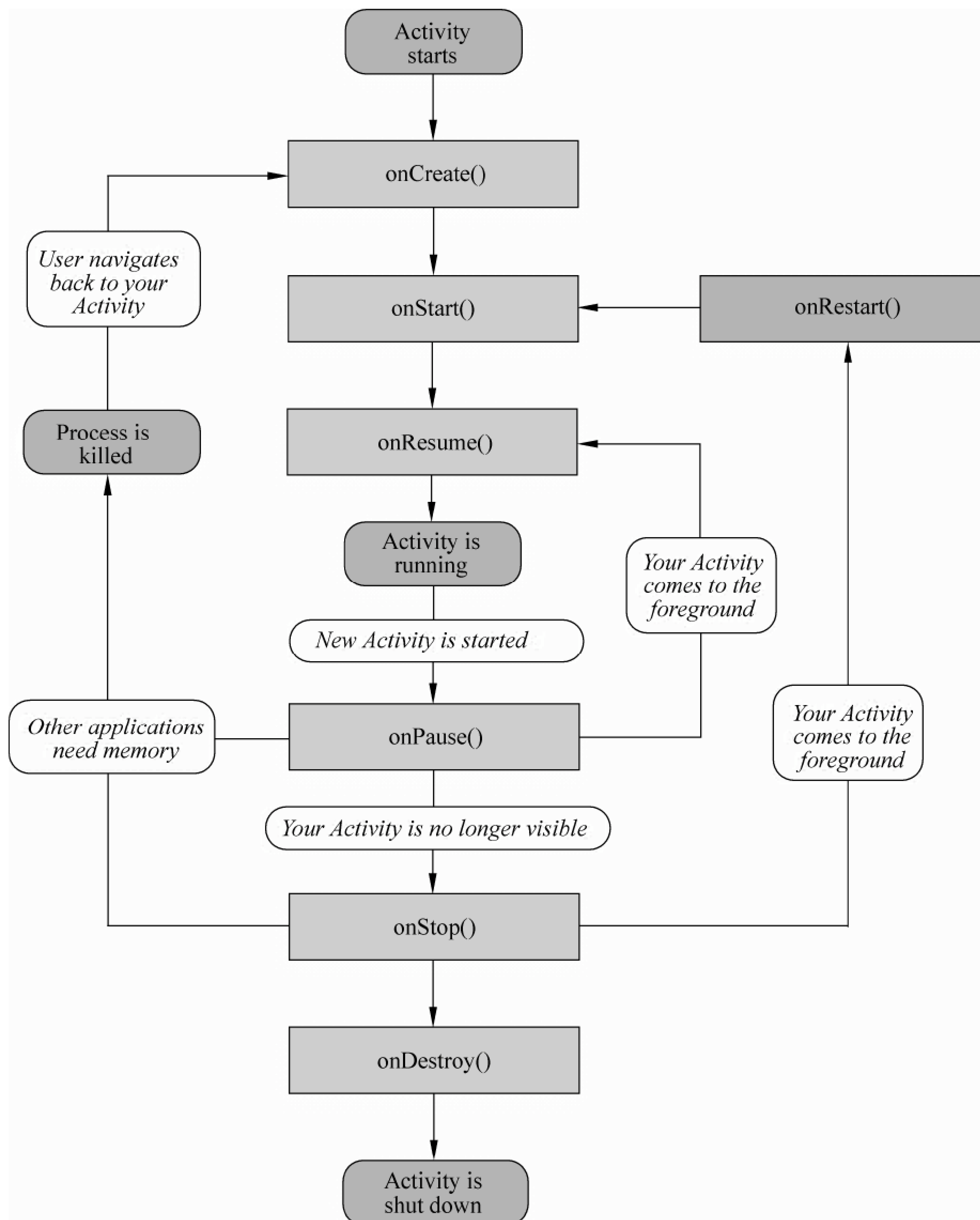


图 2.12　Activity 生命周期

首先，我们新建一个项目 ActivityTest，通过在各个继承的函数中加入 log 信息来跟踪程序执行的状况。第一个 Activity 的代码如下所示：

```
01  package com.supermario.activitytest;
02  import android.app.Activity;
03  import android.content.Intent;
04  import android.os.Bundle;
05  import android.util.Log;
06  import android.view.View;
07  import android.widget.Button;
08  public class ActivityTest extends Activity {
09      /** Called when the activity is first created. */
10      private static String TAG="ActivityTest";
```

```
11      @Override
12      //程序创建
13      public void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.main);
16          Log.e(TAG,"onCreate");
17          Button button1 = (Button) findViewById(R.id.button1);
18          // 监听 button 的事件信息
19          button1.setOnClickListener(new Button.OnClickListener() {
20              public void onClick(View v)
21              {
22                  // 新建一个 Intent 对象
23                  Intent intent = new Intent();
24                  // 指定 intent 要启动的类
25                  intent.setClass(ActivityTest.this, Activity2.class);
26                  //启动一个新的 Activity
27                  startActivity(intent);
28                  // 关闭当前的 Activity
29                  ActivityTest.this.finish();
30              }
31          });
32          Button button2 = (Button) findViewById(R.id.button2);
33          // 监听 button 的事件信息
34          button2.setOnClickListener(new Button.OnClickListener() {
35              public void onClick(View v)
36              {
37                  // 关闭当前的 Activity
38                  ActivityTest.this.finish();
39              }
40          });
41      }
42      //程序开始
43      public void onStart()
44      {
45          super.onStart();
46          Log.e(TAG,"onStart");
47      }
48      //程序恢复
49      public void onResume()
50      {
51          super.onResume();
52          Log.v(TAG, "onResume");
53      }
54      //程序暂停
55      public void onPause()
56      {
57          super.onPause();
58          Log.v(TAG, "onPause");
59      }
60      //程序停止
61      public void onStop()
62      {
63          super.onStop();
64          Log.v(TAG, "onStop");
65      }
66      //程序销毁
67      public void onDestroy()
68      {
69          super.onDestroy();
```

```
70          Log.v(TAG, "onDestroy");
71      }
72      //程序重启
73      public void onRestart()
74      {
75          super.onRestart();
76          Log.v(TAG, "onReStart");
77      }
78  }
```

第二个 Activity 的代码如下所示。与 ActivityTest 类似，在每个方法中加上一些 log 信息，用于记录当前的运行状态。

```
01  package com.supermario.activitytest;
02
03  import android.app.Activity;
04  import android.content.Intent;
05  import android.os.Bundle;
06  import android.util.Log;
07  import android.view.View;
08  import android.widget.Button;
09
10  public class Activity2 extends Activity {
11      /** Called when the activity is first created. */
12      private static String TAG="Activity2";
13      @Override
14      //程序创建
15      public void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.main2);
18          Log.e(TAG,"onCreate");
19          Button button1 = (Button) findViewById(R.id.button1);
20          //监听 button 的事件信息
21          button1.setOnClickListener(new Button.OnClickListener() {
22              public void onClick(View v)
23              {
24                  // 新建一个 Intent 对象
25                  Intent intent = new Intent();
26                  // 指定 intent 要启动的类
27                  intent.setClass(Activity2.this, ActivityTest.class);
28                  //启动一个新的 Activity
29                  startActivity(intent);
30                  //关闭当前的 Activity
31                  Activity2.this.finish();
32              }
33          });
34          Button button2 = (Button) findViewById(R.id.button2);
35          // 监听 button 的事件信息
36          button2.setOnClickListener(new Button.OnClickListener() {
37              public void onClick(View v)
38              {
39                  //关闭当前的 Activity
40                  Activity2.this.finish();
41              }
42          });
43      }
44      //程序开始
45      public void onStart()
46      {
```

```
47            super.onStart();
48            Log.e(TAG,"onStart");
49        }
50        //程序恢复
51        public void onResume()
52        {
53            super.onResume();
54            Log.v(TAG, "onResume");
55        }
56        //程序暂停
57        public void onPause()
58        {
59            super.onPause();
60            Log.v(TAG, "onPause");
61        }
62        //程序停止
63        public void onStop()
64        {
65            super.onStop();
66            Log.v(TAG, "onStop");
67        }
68        //程序销毁
69        public void onDestroy()
70        {
71            super.onDestroy();
72            Log.v(TAG, "onDestroy");
73        }
74        //程序重启
75        public void onRestart()
76        {
77            super.onRestart();
78            Log.v(TAG, "onReStart");
79        }
80    }
```

我们在布局文件中简单定义了两个 Button 和一个 TextView，一个 Button 用来切换到另一个 Activity，另一个 Button 用于关闭当前界面，TextView 用来显示当前 Activity 的名称。第一个 Activity 对应的布局文件 main.xml 代码如下：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout_width="fill_parent"
04      android:layout height="fill parent"
05      android:orientation="vertical" >
06      <!--显示当前的 Activity 标题-->
07      <TextView
08          android:layout width="fill parent"
09          android:layout height="wrap content"
10          android:text="第一个 Activity" />
11      <!--切换到第二个 Activity-->
12      <Button
13          android:id="@+id/button1"
14          android:layout width="wrap content"
15          android:layout height="wrap content"
16          android:text="切换到第二个 Activity" />
17      <!--关闭当前 Activity-->
18      <Button
19          android:id="@+id/button2"
```

```
20          android:layout width="wrap content"
21          android:layout height="wrap content"
22          android:text="关闭" />
23  </LinearLayout>
```

第二个 Activity 对应的布局文件 main2.xml 代码如下：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout_width="fill_parent"
04      android:layout_height="fill_parent"
05      android:orientation="vertical" >
06      <!--显示当前 Activity 标题-->
07      <TextView
08          android:layout width="fill parent"
09          android:layout height="wrap content"
10          android:text="第二个 Activity" />
11      <!--切换按钮-->
12      <Button
13          android:id="@+id/button1"
14          android:layout width="wrap content"
15          android:layout height="wrap content"
16          android:text="切换到第一个 Activity" />
17      <!--关闭当前 Activity-->
18      <Button
19          android:id="@+id/button2"
20          android:layout width="wrap content"
21          android:layout height="wrap content"
22          android:text="关闭" />
23  </LinearLayout>
```

然后我们要在 AndroidManifest.xml 文件中声明这两个 Activity，如下所示：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03      package="com.supermario.activitytest"
04      android:versionCode="1"
05      android:versionName="1.0" >
06      <!--声明最低的 SDK 版本-->
07      <uses-sdk android:minSdkVersion="10" />
08      <application
09          android:icon="@drawable/ic launcher"
10          android:label="@string/app name" >
11          <!--主体 Activity-->
12          <activity
13              android:name=".ActivityTest"
14              android:label="@string/app_name" >
15              <intent-filter>
16                  <action android:name="android.intent.action.MAIN" />
17
18                  <category android:name="android.intent.category.LAUNCHER" />
19              </intent-filter>
20          </activity>
21          <!--声明第二个 Activity-->
22          <activity
23              android:name=".Activity2" >
24          </activity>
25      </application>
26  </manifest>
```

最后我们运行此程序。首先我们启动程序，可以看到 Activity 经历了 onCreate()、onStart()、onResume()这 3 个函数，如图 2.13 所示。

Application	Tag	Text
com.supermario.activitytest	ActivityTest	onCreate
com.supermario.activitytest	ActivityTest	onStart
com.supermario.activitytest	ActivityTest	onResume

图 2.13　ActivityTest 启动

然后我们单击“切换到第二个 Activity”，可以看到 ActivityTest 显示执行了 onPause()，如图 2.14 所示。接下来执行 Activity2 中的 onCreate()→onStart()→onResume()，然后在 ActivityTest 中执行 onStop()→onDestroy()将 ActivityTest 依次停止、销毁。

com.supermario.activitytest	ActivityTest	onPause
com.supermario.activitytest	Activity2	onCreate
com.supermario.activitytest	Activity2	onStart
com.supermario.activitytest	Activity2	onResume
com.supermario.activitytest	ActivityTest	onStop
com.supermario.activitytest	ActivityTest	onDestroy

图 2.14　切换到 Activity2

我们在第二个 Activity 中单击“关闭”，可以看到 log 信息如图 2.15 所示，Activity 依次执行了 onPause()→onStop()→onDestroy()。

com.supermario.activitytest	Activity2	onPause
com.supermario.activitytest	Activity2	onStop
com.supermario.activitytest	Activity2	onDestroy

图 2.15　单击“关闭”

2.4　本 章 小 结

本章主要介绍了 Android 四大组件及 Activity 的生命周期，并通过 HelloAndroid 这个项目分析了 Android 的目录结构、文件功能等。通过一个个功能简单的小程序，让大家明白 Android 应用程序其实就是由这一点点的小功能组成的，理解每一个功能，可以帮助你编写一个高效正确的 Android 应用程序。

第 2 篇　Android 典型应用实战案例

第3章　计　算　器

前两章我们学习了Android开发的一些基础知识，学习了如何用Eclipse来创建工程，学习了调试程序的方法。这章我们将编写一个计算器，计算器的功能可以很简单，只有加减乘除，也可以很复杂，带科学计算功能等。

3.1　功 能 分 析

计算器有很多种，各行各业用的、简单的、复杂的，作为学习我们选择常用的科学计算器作为例子。读者若有兴趣开发其他计算器，只要按照这个计算器的模板增加按键和相应的算法即可。计算器的具体功能就不再赘述了，直接看一下我们要做成的效果图吧，如图 3.1 所示。



图 3.1　计算器界面设计

3.2　界 面 设 计

计算器的界面属于方方正正的那种，考虑到布局方便和界面美观，我们采用TableRow。

每一行是一个TableRow，TableRow中每放一个元素就是一列，总列数由列数最多的那一行决定。

如下所示，我们模仿现实中的计算器，定义了一个EditText用于显示输出结果，定义了一个TextView用于显示提示。这里提示的作用相当于使用帮助，比如计算完毕之后会在这里显示“计算完毕要继续请按归零键C”，如果出错了，会提示相应的错误原因。其余的按键我们都用Button定义。

```
001 <?xml version="1.0" encoding="utf-8"?>
002 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
003    android:orientation="vertical"
004    android:layout_width="fill_parent"
005    android:layout_height="fill_parent"
006    <!-- 设置背景色为灰色 -->
007    android:background="#ff808080"
008    >
009    <!-- 结果显示框 -->
010    <EditText android:id="@+id/input"
011        android:layout_width="fill_parent"
012        android:layout_height="wrap_content"
013        <!-- 光标不可见 -->
014        android:cursorVisible="false"
015        <!-- 向右对齐 -->
016        android:gravity="right"
017        <!-- 不可编辑 -->
018        android:editable = "false"
019        <!-- 默认显示字符串"0"-->
020        android:text="0"                              />
021    <!-- 接下去采用 TableRow 的格式进行布局设计 -->
022    <TableRow
023         android:layout_width="fill_parent"
024         android:layout_height="wrap_content"
025         >
026            <!-- 用于显示存储结果 -->
027            <TextView android:id="@+id/M"
028                android:layout_width="53sp"
029                android:layout_height="wrap_content"
030                android:text="  MEM :"                 />
031            <!-- 默认显示字符串"0" -->
032            <TextView android:id="@+id/mem"
033                android:layout_width="fill_parent"
034                android:layout_height="wrap_content"
035                android:text="0"                      />
036    </TableRow>
037    <TableRow
038         android:layout_width="fill_parent"
039         android:layout_height="wrap_content"
040         >
041            <!-- 显示当前是角度还是弧度，默认是角度 -->
042            <TextView android:id="@+id/ drg"
043                android:layout_width="53sp"
044                android:layout_height="wrap_content"
045                android:text="   DEG"                  />
046            <!-- 清除存储结果 -->
047            <Button android:id="@+id/mc"
048                android:text="MC"
```

```
049                 android:layout_width="106sp"
050                 android:layout_height="wrap_content"   />
051              <!-- 清除输出窗口的所有内容 -->
052             <Button android:id="@+id/c"
053              android:text="C"
054                 android:layout_width="fill_parent"
055                 android:layout_height="wrap_content"   />
056     </TableRow>
057     <TableRow
058         android:layout_width="fill_parent"
059         android:layout_height="wrap_content"
060         >
061             <!-- 在角度和弧度之间切换 -->
062             <Button android:id="@+id/drg"
063              android:text="DRG"
064                 android:layout_width="53sp"
065                 android:layout_height="wrap_content" />
066             <!-- 正弦计算 -->
067             <Button android:id="@+id/sin"
068              android:text="sin"
069                 android:layout_width="53sp"
070                 android:layout_height="wrap_content" />
071             <!-- 余弦计算 -->
072             <Button android:id="@+id/cos"
073              android:text="cos"
074                 android:layout_width="53sp"
075                 android:layout_height="wrap_content" />
076             <!-- 正切计算 -->
077             <Button android:id="@+id/tan"
078              android:text="tan"
079                 android:layout_width="53sp"
080                 android:layout_height="wrap_content" />
081             <!-- 阶乘计算 -->
082             <Button android:id="@+id/factorial"
083              android:text="n!"
084                 android:layout_width="53sp"
085                 android:layout_height="wrap_content" />
086             <!-- 退格键 -->
087             <Button android:id="@+id/bksp"
088              android:text="Bksp"
089                 android:layout_width="53sp"
090                 android:layout_height="wrap_content" />
091     </TableRow>
092     <TableRow
093         android:layout_width="fill_parent"
094         android:layout_height="wrap_content"
095         >
096             <!-- 数字 7 -->
097             <Button android:id="@+id/seven"
098              android:text="7"
099                 android:layout_width="53sp"
100                 android:layout_height="wrap_content" />
101             <!-- 数字 8 -->
102             <Button android:id="@+id/eight"
103              android:text="8"
104                 android:layout_width="53sp"
105                 android:layout_height="wrap_content" />
106             <!-- 数字 9 -->
107             <Button android:id="@+id/nine"
108              android:text="9"
```

```
109                 android:layout_width="53sp"
110                 android:layout_height="wrap_content" />
111             <!-- 除号 -->
112             <Button android:id="@+id/divide"
113              android:text="÷"
114                 android:layout_width="53sp"
115                 android:layout_height="wrap_content" />
116             <!-- 左括号 -->
117             <Button android:id="@+id/left"
118              android:text="("
119                 android:layout_width="53sp"
120                 android:layout_height="wrap_content" />
121             <!-- 右括号 -->
122             <Button android:id="@+id/right"
123              android:text=")"
124                 android:layout_width="53sp"
125                 android:layout_height="wrap_content" />
126     </TableRow>
127     <TableRow
128         android:layout_width="fill_parent"
129         android:layout_height="wrap_content"
130         >
131             <!-- 数字 4 -->
132             <Button android:id="@+id/four"
133              android:text="4"
134                 android:layout_width="53sp"
135                 android:layout_height="wrap_content" />
136             <!-- 数字 5 -->
137             <Button android:id="@+id/five"
138              android:text="5"
139                 android:layout_width="53sp"
140                 android:layout_height="wrap_content" />
141             <!-- 数字 6 -->
142             <Button android:id="@+id/six"
143              android:text="6"
144                 android:layout_width="53sp"
145                 android:layout_height="wrap_content" />
146             <!-- 乘号 -->
147             <Button android:id="@+id/mul"
148              android:text="×"
149                 android:layout_width="53sp"
150                 android:layout_height="wrap_content" />
151             <!-- 开方 -->
152             <Button android:id="@+id/sqrt"
153              android:text="√"
154                 android:layout_width="53sp"
155                 android:layout_height="wrap_content" />
156             <!-- 乘方 -->
157             <Button android:id="@+id/square"
158              android:text="^"
159                 android:layout_width="53sp"
160                 android:layout_height="wrap_content" />
161     </TableRow>
162     <TableRow
163         android:layout_width="fill_parent"
164         android:layout_height="wrap_content"
165         >
166             <!-- 数字 1 -->
167             <Button android:id="@+id/one"
168              android:text="1"
```

```
169                android:layout_width="53sp"
170                android:layout_height="wrap_content" />
171             <!-- 数字 2 -->
172             <Button android:id="@+id/two"
173              android:text="2"
174                android:layout_width="53sp"
175                android:layout_height="wrap_content" />
176             <!-- 数字 3 -->
177             <Button android:id="@+id/three"
178              android:text="3"
179                android:layout_width="53sp"
180                android:layout_height="wrap_content" />
181             <!-- 减号 -->
182             <Button android:id="@+id/sub"
183              android:text="-"
184                android:layout_width="53sp"
185                android:layout_height="wrap_content" />
186             <!-- 对数 -->
187             <Button android:id="@+id/log"
188              android:text="log"
189                android:layout_width="53sp"
190                android:layout_height="wrap_content" />
191             <!-- 自然对数 -->
192             <Button android:id="@+id/ln"
193              android:text="ln"
194                android:layout_width="53sp"
195                android:layout_height="wrap_content" />
196       </TableRow>
197         <TableLayout xmlns:android="http://schemas.android.com/apk/
         res/android"
198           android:layout_width="fill_parent"
199           android:layout_height="57sp"
200         >
201       <TableRow
202           android:layout_width="fill_parent"
203           android:layout_height="wrap_content"
204           >
205             <!-- 数字 0 -->
206             <Button android:id="@+id/zero"
207              android:text="0"
208                android:layout_width="53sp"
209                android:layout_height="wrap_content" />
210             <!-- 小数点 -->
211             <Button android:id="@+id/dot"
212              android:text="."
213                android:layout_width="53sp"
214                android:layout_height="wrap_content" />
215             <!-- 等号 -->
216             <Button android:id="@+id/equal"
217              android:text="="
218                android:layout_width="53sp"
219                android:layout_height="wrap_content" />
220             <!-- 加号 -->
221             <Button android:id="@+id/add"
222              android:text="+"
223                android:layout_width="53sp"
224                android:layout_height="wrap_content" />
225             <!-- 退出计算器 -->
226             <Button android:id="@+id/exit"
227              android:text="exit"
```

```
228                 android:layout width="106sp"
229                 android:layout height="wrap content" />
230       </TableRow>
231       </TableLayout>
232       <TableRow
233            android:layout width="fill parent"
234            android:layout height="wrap content"
235            >
236              <!-- 用于提示，告诉用户如何使用计算器的一些功能等 -->
237              <TextView android:id="@+id/T"
238                 android:layout_width="45sp"
239                 android:layout height="wrap content"
240                 android:text="提示: "              />
241              <TextView android:id="@+id/tip"
242                 android:layout width="fill parent"
243                 android:layout height="wrap content"
244                 android:text="欢迎使用！"              />
245       </TableRow>
246<LinearLayout>
```

3.3　功 能 实 现

计算器最核心的功能就是运算，而与用户接触最紧密的就是按键和显示，因此我们将从这几方面去考虑计算器功能的具体实现。

3.3.1　定义变量

首先，最基本的我们要先定义一些变量，用来表示这些按键和文本内容，如数字按键0～9，＋、－、×、÷按键等，还有用于显示输出结果的显示器，用于显示记忆内容的文本框，用于显示提示的文本框等。定义完了按键之后，我们需要在onCreate()函数中将这些变量与main.xml中的视图绑定，同时为所有的按键绑定按键监听器actionPerformed。

```
01  //0~9 十个按键
02      private Button[] btn = new Button[10];
03      //显示器，用于显示输出结果
04      private EditText input;
05      //显示器下方的记忆器，用于记录上一次计算结果
06      private TextView mem;
07      //三角计算时标志显示：角度还是弧度
08      private TextView  drg;
09      //小提示，用于加强人机交互的弱检测、提示
10      private TextView tip;
11      private Button
12          div, mul, sub, add, equal,       // ÷、×、－、+、=
13          sin, cos, tan, log, ln,          //函数
14          sqrt, square, factorial, bksp,   //根号, 平方, 阶乘, 退格
15          left, right, dot, exit, drg,     //(, ), . , 退出, 角度弧度控制键
16          mc, c;                           // mem 清屏键, input 清屏键
17      //保存原来的算式样子，为了输出时好看，因计算时算式样子被改变
18      public String str_old;
```

```
19      //变换样子后的式子
20      public String str new;
21      //输入控制，true 为重新输入，false 为接着输入
22      public boolean vbegin = true;
23      //控制 DRG 按键，true 为角度，false 为弧度
24      public boolean drg flag = true;
25      //π 值：3.14
26      public double pi=4*Math.atan(1);
27      //true 表示正确，可以继续输入；false 表示有误，输入被锁定
28      public boolean tip lock = true;
29      //判断是否是按=之后的输入，true 表示输入在=之前，false 反之
30      public boolean equals_flag = true;
31
32      public void onCreate(Bundle icicle)
33      {
34          super.onCreate(icicle);
35          setContentView(R.layout.main);
36          //获取界面元素
37          input = (EditText)findViewById(R.id.input);
38          mem = (TextView)findViewById(R.id.mem);
39          tip = (TextView)findViewById(R.id.tip);
40           drg = (TextView)findViewById(R.id. drg);
41          btn[0] = (Button)findViewById(R.id.zero);
42           btn[1] = (Button)findViewById(R.id.one);
43           btn[2] = (Button)findViewById(R.id.two);
44           btn[3] = (Button)findViewById(R.id.three);
45           btn[4] = (Button)findViewById(R.id.four);
46           btn[5] = (Button)findViewById(R.id.five);
47           btn[6] = (Button)findViewById(R.id.six);
48           btn[7] = (Button)findViewById(R.id.seven);
49          btn[8] = (Button)findViewById(R.id.eight);
50          btn[9] = (Button)findViewById(R.id.nine);
51          div = (Button)findViewById(R.id.divide);
52          mul = (Button)findViewById(R.id.mul);
53          sub = (Button)findViewById(R.id.sub);
54          add = (Button)findViewById(R.id.add);
55          equal = (Button)findViewById(R.id.equal);
56          sin = (Button)findViewById(R.id.sin);
57          cos = (Button)findViewById(R.id.cos);
58          tan = (Button)findViewById(R.id.tan);
59          log = (Button)findViewById(R.id.log);
60          ln = (Button)findViewById(R.id.ln);
61          sqrt = (Button)findViewById(R.id.sqrt);
62          square = (Button)findViewById(R.id.square);
63          factorial = (Button)findViewById(R.id.factorial);
64          bksp = (Button)findViewById(R.id.bksp);
65          left = (Button)findViewById(R.id.left);
66          right = (Button)findViewById(R.id.right);
67          dot = (Button)findViewById(R.id.dot);
68          exit = (Button)findViewById(R.id.exit);
69          drg = (Button)findViewById(R.id.drg);
70          mc = (Button)findViewById(R.id.mc);
71          c = (Button)findViewById(R.id.c);
72          //为数字按键绑定监听器
73          for(int i = 0; i < 10; ++i) {
74              btn[i].setOnClickListener(actionPerformed);
75          }
76          //为+、-、x、÷等按键绑定监听器
77          div.setOnClickListener(actionPerformed);
78          mul.setOnClickListener(actionPerformed);
```

```
79          sub.setOnClickListener(actionPerformed);
80          add.setOnClickListener(actionPerformed);
81          equal.setOnClickListener(actionPerformed);
82          sin.setOnClickListener(actionPerformed);
83          cos.setOnClickListener(actionPerformed);
84          tan.setOnClickListener(actionPerformed);
85          log.setOnClickListener(actionPerformed);
86          ln.setOnClickListener(actionPerformed);
87          sqrt.setOnClickListener(actionPerformed);
88          square.setOnClickListener(actionPerformed);
89          factorial.setOnClickListener(actionPerformed);
90          bksp.setOnClickListener(actionPerformed);
91          left.setOnClickListener(actionPerformed);
92          right.setOnClickListener(actionPerformed);
93          dot.setOnClickListener(actionPerformed);
94          exit.setOnClickListener(actionPerformed);
95          drg.setOnClickListener(actionPerformed);
96          mc.setOnClickListener(actionPerformed);
97          c.setOnClickListener(actionPerformed);
98      }
```

3.3.2　actionPerformed()函数

从上面的代码可以看出这个程序的关键就是actionPerformed()函数，下面我们来看一下这个函数的实现。跟其他监听器一样，这个监听器的核心也是实现onClick方法。

程序的前面定义了变量Tipcommand用于缓存命令，在检测输入是否合法时需要用到这个变量。流程一开始在获取了按键之后，先用函数right检测当前显示器中的字符串是否合法，如果合法则将字符串缓存到Tipcommand中，否则将显示器置为 0。

接下去将缓存的字符串的最后一位（若当前传入的不含二元运算符，则传入“#”）和当前输入命令一起输入到函数TipChecker中进行分析，检查输入的合法性。如果输入的是数字或者运算符，就将新输入的命令增加到缓存的字符串Tipcommand中，并显示到显示器上。如果输入的是角度弧度切换按钮，则将相应标志位改变，并将当前是角度或者弧度显示出来。如果输入的是退格键bksp，则先通过函数TTO判断是要删掉几个字符串，有可能是一个、两个，也有可能是三个。如果当前标志是已经输入过“=”的标志位equal_flag为false，表明已经输入过“=”，此时如果输入退格键，则将显示器清零。如果是MC，则清除存储器内容；如果是C，则清除显示器内容；如果是Exit，则退出计算器。

如果输入的是“=”，则进入计算器的核心功能——计算了。首先设置一些标志位，如tip_i=0、tip_lock=false、vbegin=false，然后保存当前显示器的字符串，并替换当前字符串中一些函数的名字，以便于计算操作。执行类calc中的函数process进行计算。

整个程序的流程都在这个actionPerformed中实现，这其中用到了一些函数，我们接下去来分析一下。

```
001 /*
002      * 键盘命令捕捉
003      */
004     //命令缓存，用于检测输入合法性
005     String[] Tipcommand = new String[500];
006     //Tipcommand 的指针
007     int tip_i = 0;
```

```
008    private OnClickListener actionPerformed = new OnClickListener() {
009        public void onClick(View v) {
010         //按键上的命令获取
011            String command = ((Button)v).getText().toString();
012            //显示器上的字符串
013            String str = input.getText().toString();
014            //检测输入是否合法
015            if(equals flag == false && "0123456789.()sincostanlnlogn
               !+-×÷√^".indexOf(command) != -1) {
016             //检测显示器上的字符串是否合法
017                if(right(str)) {
018                    if("+-×÷√^)".indexOf(command) != -1) {
019                        for(int i =0 ; i < str.length(); i++) {
020                            Tipcommand[tip i] = String.valueOf(str
                               .charAt(i));
021                            tip i++;
022                        }
023                        vbegin = false;
024                    }
025                } else {
026                    input.setText("0");
027                    vbegin = true;
028                    tip i = 0;
029                    tip lock = true;
030                    tip.setText("欢迎使用！");
031                }
032
033                equals flag = true;
034            }
035            if(tip i > 0)
036                TipChecker(Tipcommand[tip_i-1] , command);
037            else if(tip i == 0) {
038                    TipChecker("#" , command);
039            }
040            if("0123456789.()sincostanlnlogn!+-×÷√^".indexOf(command)
               != -1 && tip lock) {
041                Tipcommand[tip i] = command;
042                tip i++;
043            }
044            //若输入正确，则将输入信息显示到显示器上
045            if("0123456789.()sincostanlnlogn!+-×÷√^".indexOf(command)
               != -1
046                    && tip lock) { //共 25 个按键
047                print(command);
048            //如果单击了 DRG，则切换当前弧度角度制，并将切换后的结果显示到按键上方
049            } else if(command.compareTo("DRG") == 0 && tip_lock) {
050                if(drg_flag == true) {
051                    drg_flag = false;
052                    _drg.setText("   RAD");
053                } else {
054                    drg flag = true;
055                     drg.setText("   DEG");
056                }
057                //如果输入是退格键，并且是在按=之前
058            } else if(command.compareTo("Bksp") == 0 && equals_flag) {
059             //一次删除 3 个字符
060                if(TTO(str) == 3) {
061                    if(str.length() > 3)
062                        input.setText(str.substring(0, str.length() - 3));
063                    else if(str.length() == 3) {
```

```
064                     input.setText("0");
065                     vbegin = true;
066                     tip_i = 0;
067                     tip.setText("欢迎使用！");
068                 }
069             //依次删除 2 个字符
070             } else if(TTO(str) == 2) {
071                 if(str.length() > 2)
072                     input.setText(str.substring(0, str.length() - 2));
073                 else if(str.length() == 2) {
074                     input.setText("0");
075                     vbegin = true;
076                     tip_i = 0;
077                     tip.setText("欢迎使用！");
078                 }
079             //依次删除一个字符
080             } else if(TTO(str) == 1) {
081              //若之前输入的字符串合法，则删除一个字符
082                 if(right(str)) {
083                     if(str.length() > 1)
084                         input.setText(str.substring(0, str.length() - 1));
085                     else if(str.length() == 1) {
086                         input.setText("0");
087                         vbegin = true;
088                         tip_i = 0;
089                         tip.setText("欢迎使用！");
090                     }
091                     //若之前输入的字符串不合法，则删除全部字符
092                 } else {
093                     input.setText("0");
094                     vbegin = true;
095                     tip_i = 0;
096                     tip.setText("欢迎使用！");
097                 }
098             }
099             if(input.getText().toString().compareTo("-") == 0 ||
                equals_flag == false) {
100                 input.setText("0");
101                 vbegin = true;
102                 tip_i = 0;
103                 tip.setText("欢迎使用！");
104             }
105             tip_lock = true;
106             if(tip_i > 0)
107                 tip_i--;
108         //如果是在按=之后输入退格键
109         }else if(command.compareTo("Bksp")==0 && equals_flag==false){
110          //将显示器内容设置为 0
111             input.setText("0");
112             vbegin = true;
113             tip_i = 0;
114             tip_lock = true;
115             tip.setText("欢迎使用！");
116         //如果输入的是清除键
117         } else if(command.compareTo("C") == 0) {
118          //将显示器内容设置为 0
119             input.setText("0");
120             //重新输入标志置为 true
```

```
121                vbegin = true;
122                //缓存命令位数清 0
123                tip_i = 0;
124                //表明可以继续输入
125                tip_lock = true;
126                //表明输入=之前
127                equals_flag = true;
128                tip.setText("欢迎使用！");
129                //如果输入的是 MC，则将存储器内容清 0
130            } else if(command.compareTo("MC") == 0) {
131                mem.setText("0");
132                //如果按 Exit 则退出程序
133            } else if(command.compareTo("exit") == 0) {
134                System.exit(0);
135            //如果输入的是=号，并且输入合法
136            } else if(command.compareTo("=") == 0 && tip_lock && right(str)
            && equals_flag) {
137                tip_i = 0;
138                //表明不可以继续输入
139                tip_lock = false;
140                //表明输入=之后
141                equals_flag = false;
142                //保存原来算式的样子
143                str_old = str;
144                //替换算式中的运算符，便于计算
145                str = str.replaceAll("sin", "s");
146                str = str.replaceAll("cos", "c");
147                str = str.replaceAll("tan", "t");
148                str = str.replaceAll("log", "g");
149                str = str.replaceAll("ln", "l");
150                str = str.replaceAll("n!", "!");
151                //重新输入标志设置为 true
152                vbegin = true;
153                //将-1x 转换成-
154                str_new = str.replaceAll("-", "-1×");
155                //计算算式结果
156                new calc().process(str_new);
157            }
158            //表明可以继续输入
159            tip_lock = true;
160        }
161    };
```

3.3.3　print()函数

首先是print()函数，这个函数根据变量vbegin来判断是在当前显示内容后面追加字符串还是清零后显示。然后是right()函数，这个函数通过判断字符串内容是否只包含“0123456789+-x÷.()sincostanlnlog√^”来判断当前字符串是否合法，合法返回true，不合法则返回false。最后是TTO()函数，判断当前字符串最后一个字符的内容来决定是删除一个、两个还是三个字符串。

```
01   //向 input 输出字符
02    private void print(String str) {
03      //清屏后输出
```

```
04        if(vbegin)
05            input.setText(str);
06        //在屏幕原 str 后增添字符
07        else
08            input.append(str);
09        vbegin = false;
10    }
11    /*
12     * 判断一个 str 是否是合法的，返回值为 true、false
13     * 只包含 0123456789.()sincostanlnlogn!+-×÷√^的是合法的 str，返回 true
14     * 包含了除 0123456789.()sincostanlnlogn!+-×÷√^以外的字符的 str 为非法的，
       返回 false
15     */
16    private boolean right(String str) {
17        int i = 0;
18        for(i = 0;i < str.length();i++) {
19            if(str.charAt(i)!='0' && str.charAt(i)!='1' && str.charAt(i)!=
              '2' &&
20                     str.charAt(i)!='3' && str.charAt(i)!='4' && str.charAt
                      (i)!='5' &&
21                     str.charAt(i)!='6' && str.charAt(i)!='7' && str.charAt
                      (i)!='8' &&
22                     str.charAt(i)!='9' && str.charAt(i)!='.' && str.charAt
                      (i)!='-' &&
23                     str.charAt(i)!='+' && str.charAt(i)!='×' && str.charAt
                      (i)!='÷' &&
24                     str.charAt(i)!='√' && str.charAt(i)!='^' && str.charAt
                      (i)!='s' &&
25                     str.charAt(i)!='i' && str.charAt(i)!='n' && str.charAt
                      (i)!='c' &&
26                     str.charAt(i)!='o' && str.charAt(i)!='t' && str.charAt
                      (i)!='a' &&
27                     str.charAt(i)!='l' && str.charAt(i)!='g' && str.charAt
                      (i)!='(' &&
28                     str.charAt(i)!=')' && str.charAt(i)!='!')
29                break;
30        }
31        if(i == str.length()) {
32            return true;
33        } else {
34            return false;
35        }
36    }
37    /*
38     * 检测函数，返回值为 3、2、1，表示应当一次删除几个 TTO（即 Three Two One）为
       函数名
39     * 为 Bksp 按钮的删除方式提供依据
40     * 返回 3，表示 str 尾部为 sin、cos、tan、log 中的一个，应当一次删除 3 个
41     * 返回 2，表示 str 尾部为 ln、n!中的一个，应当一次删除 2 个
42     * 返回 1，表示为除返回 3、2 外的所有情况，只需删除一个（包含非法字符时要另外考虑：
       应清屏）
43     */
44    private int TTO(String str) {
45        if((str.charAt(str.length() - 1) == 'n' &&
46                str.charAt(str.length() - 2) == 'i' &&
47                str.charAt(str.length() - 3) == 's') ||
48                (str.charAt(str.length() - 1) == 's' &&
49                str.charAt(str.length() - 2) == 'o' &&
```

```
50                str.charAt(str.length() - 3) == 'c') ||
51                (str.charAt(str.length() - 1) == 'n' &&
52                str.charAt(str.length() - 2) == 'a' &&
53                str.charAt(str.length() - 3) == 't') ||
54                (str.charAt(str.length() - 1) == 'g' &&
55                str.charAt(str.length() - 2) == 'o' &&
56                str.charAt(str.length() - 3) == 'l')) {
57            return 3;
58        } else if((str.charAt(str.length() - 1) == 'n' &&
59                       str.charAt(str.length() - 2) == 'l') ||
60                       (str.charAt(str.length() - 1) == '!' &&
61                       str.charAt(str.length() - 2) == 'n')) {
62                   return 2;
63        } else { return 1; }
64    }
```

3.3.4　TipChecker()函数

TipChecker()这个函数在整个程序中很重要，主要有两个作用，一是用来检测当前输入的字符串是否合法，二是对某些函数的使用显示一些帮助信息。

函数的一开始定义了错误类型和按键类型，接着一方面枚举每种可能出现错误的情况，将这些错误类型以数字的形式表示出来；另一方面，若输入的是sin、cos之类，将会在最底下显示提示信息，以指导用户使用这些函数，从某种角度上来说是一种预防错误的措施。

最后通过函数TipShow()将信息显示出来。

```
001    /*
002     * 检测函数，对 str 进行前后语法检测
003     * 为 Tip 的提示方式提供依据，与 TipShow()配合使用
004     *  编号 字符     其后可以跟随的合法字符
005     *   1   (             数字|(|-|.|函数
006     *   2   )             算符|)|√ ^
007     *   3  .              数字|算符|)|√ ^
008     *   4   数字           .|数字|算符|)|√ ^
009     *   5   算符           数字|(|.|函数
010     *   6 √ ^              ( |. | 数字
011     *   7  函数            数字|(|.
012     *
013     * 小数点前后均可省略，表示 0
014     * 数字第一位可以为 0
015     */
016    private void TipChecker(String tipcommand1,String tipcommand2) {
017     //Tipcode1 表示错误类型，Tipcode2 表示名词解释类型
018     int Tipcode1 = 0 , Tipcode2 = 0;
019       //表示命令类型
020       int tiptype1 = 0 , tiptype2 = 0;
021       //括号数
022       int bracket = 0;
023       //"+-x÷√^"不能作为第一位
024       if(tipcommand1.compareTo("#") == 0 && (tipcommand2.compareTo
          ("÷") == 0 ||
025            tipcommand2.compareTo("×") == 0 || tipcommand2.compareTo
```

```
                ("+") == 0 ||
026             tipcommand2.compareTo(")") == 0 || tipcommand2.compareTo
                ("√") == 0 ||
027             tipcommand2.compareTo("^") == 0)) {
028         Tipcode1 = -1;
029     }
030     //定义存储字符串中最后一位的类型
031     else if(tipcommand1.compareTo("#") != 0) {
032         if(tipcommand1.compareTo("(") == 0) {
033             tiptype1 = 1;
034         } else if(tipcommand1.compareTo(")") == 0) {
035             tiptype1 = 2;
036         } else if(tipcommand1.compareTo(".") == 0) {
037             tiptype1 = 3;
038         } else if("0123456789".indexOf(tipcommand1) != -1) {
039             tiptype1 = 4;
040         } else if("+-×÷".indexOf(tipcommand1) != -1) {
041             tiptype1 = 5;
042         } else if("√^".indexOf(tipcommand1) != -1) {
043             tiptype1 = 6;
044         } else if("sincostanloglnn!".indexOf(tipcommand1) != -1) {
045             tiptype1 = 7;
046         }
047         //定义欲输入的按键类型
048         if(tipcommand2.compareTo("(") == 0) {
049             tiptype2 = 1;
050         } else if(tipcommand2.compareTo(")") == 0) {
051             tiptype2 = 2;
052         } else if(tipcommand2.compareTo(".") == 0) {
053             tiptype2 = 3;
054         } else if("0123456789".indexOf(tipcommand2) != -1) {
055             tiptype2 = 4;
056         } else if("+-×÷".indexOf(tipcommand2) != -1) {
057             tiptype2 = 5;
058         } else if("√^".indexOf(tipcommand2) != -1) {
059             tiptype2 = 6;
060         } else if("sincostanloglnn!".indexOf(tipcommand2) != -1) {
061             tiptype2 = 7;
062         }
063
064         switch(tiptype1) {
065         case 1:
066          //左括号后面直接接右括号,"+x÷"（负号"-"不算）,或者"√^"
067             if(tiptype2 == 2 || (tiptype2 == 5 && tipcommand2.compareTo
                ("-") != 0) ||
068                     tiptype2 == 6)
069                 Tipcode1 = 1;
070             break;
071         case 2:
072          //右括号后面接左括号、数字、"+-x÷sin^..."
073             if(tiptype2 == 1 || tiptype2 == 3 || tiptype2 == 4 || tiptype2
                == 7)
074              Tipcode1 = 2;
075             break;
076         case 3:
077          //"."后面接左括号或者"sincos..."
078             if(tiptype2 == 1 || tiptype2 == 7)
079              Tipcode1 = 3;
080             //连续输入两个"."
081             if(tiptype2 == 3)
```

```
082                 Tipcode1 = 8;
083                break;
084            case 4:
085             //数字后面直接接左括号或者"sincos..."
086                if(tiptype2 == 1 || tiptype2 == 7)
087                 Tipcode1 = 4;
088                break;
089            case 5:
090             //"+-x÷"后面直接接右括号、"+-x÷√^"
091                if(tiptype2 == 2 || tiptype2 == 5 || tiptype2 == 6)
092                 Tipcode1 = 5;
093                break;
094            case 6:
095             //"√^"后面直接接右括号、"+-x÷√^"以及"sincos... "
096                if(tiptype2 == 2 || tiptype2 == 5 || tiptype2 == 6 || tiptype2
                   == 7)
097                 Tipcode1 = 6;
098                break;
099            case 7:
100             //"sincos..."后面直接接右括号"+-x÷√^"以及"sincos... "
101                if(tiptype2 == 2 || tiptype2 == 5 || tiptype2 == 6 || tiptype2
                   == 7)
102                 Tipcode1 = 7;
103                break;
104            }
105        }
106        //检测小数点的重复性，Tipconde1=0，表明满足前面的规则
107        if(Tipcode1 == 0 && tipcommand2.compareTo(".") == 0) {
108            int tip_point = 0;
109            for(int i = 0;i < tip i;i++) {
110             //若之前出现一个小数点，则小数点计数加 1
111                if(Tipcommand[i].compareTo(".") == 0) {
112                    tip point++;
113                }
114                //若出现以下几个运算符之一，小数点计数清零
115                if(Tipcommand[i].compareTo("sin") == 0 || Tipcommand[i].
                   compareTo("cos") == 0 ||
116                        Tipcommand[i].compareTo("tan") == 0 || Tipcommand
                           [i].compareTo("log") == 0 ||
117                        Tipcommand[i].compareTo("ln") == 0 || Tipcommand[i].
                           compareTo("n!") == 0 ||
118                        Tipcommand[i].compareTo("√") == 0 || Tipcommand[i].
                           compareTo("^") == 0 ||
119                        Tipcommand[i].compareTo("÷") == 0 || Tipcommand[i].
                           compareTo("×") == 0 ||
120                        Tipcommand[i].compareTo("-") == 0 || Tipcommand[i].
                           compareTo("+") == 0 ||
121                        Tipcommand[i].compareTo("(") == 0 || Tipcommand[i].
                           compareTo(")") == 0 ) {
122                    tip_point = 0;
123                }
124            }
125            tip_point++;
126            //若小数点计数大于 1，表明小数点重复了
127            if(tip_point > 1) {
128                Tipcode1 = 8;
129            }
130        }
131        //检测右括号是否匹配
132        if(Tipcode1 == 0 && tipcommand2.compareTo(")") == 0) {
```

```
133             int tip_right_bracket = 0;
134             for(int i = 0;i < tip_i;i++) {
135              //如果出现一个左括号，则计数加 1
136                 if(Tipcommand[i].compareTo("(") == 0) {
137                     tip_right_bracket++;
138                 }
139                 //如果出现一个右括号，则计数减 1
140                 if(Tipcommand[i].compareTo(")") == 0) {
141                     tip_right_bracket--;
142                 }
143             }
144             //如果右括号计数=0，表明没有相应的左括号与当前右括号匹配
145             if(tip_right_bracket == 0) {
146                 Tipcode1 = 10;
147             }
148         }
149         //检查输入=的合法性
150         if(Tipcode1 == 0 && tipcommand2.compareTo("=") == 0) {
151          //括号匹配数
152          int tip_bracket = 0;
153             for(int i = 0;i < tip_i;i++) {
154                 if(Tipcommand[i].compareTo("(") == 0) {
155                     tip_bracket++;
156                 }
157                 if(Tipcommand[i].compareTo(")") == 0) {
158                     tip_bracket--;
159                 }
160             }
161             //若大于 0，表明左括号还有未匹配的
162             if(tip_bracket > 0) {
163                 Tipcode1 = 9;
164                 bracket = tip_bracket;
165             } else if(tip_bracket == 0) {
166              //若前一个字符是以下之一，表明=号不合法
167                 if("√^sincostanloglnn!".indexOf(tipcommand1) != -1) {
168                     Tipcode1 = 6;
169                 }
170                 //若前一个字符是以下之一，表明=号不合法
171                 if("+-×÷".indexOf(tipcommand1) != -1) {
172                     Tipcode1 = 5;
173                 }
174             }
175         }
176         //若命令是以下之一，则显示相应的帮助信息
177         if(tipcommand2.compareTo("MC") == 0) Tipcode2 = 1;
178         if(tipcommand2.compareTo("C") == 0) Tipcode2 = 2;
179         if(tipcommand2.compareTo("DRG") == 0) Tipcode2 = 3;
180         if(tipcommand2.compareTo("Bksp") == 0) Tipcode2 = 4;
181         if(tipcommand2.compareTo("sin") == 0) Tipcode2 = 5;
182         if(tipcommand2.compareTo("cos") == 0) Tipcode2 = 6;
183         if(tipcommand2.compareTo("tan") == 0) Tipcode2 = 7;
184         if(tipcommand2.compareTo("log") ==0) Tipcode2 = 8;
185         if(tipcommand2.compareTo("ln") == 0) Tipcode2 = 9;
186         if(tipcommand2.compareTo("n!") == 0) Tipcode2 = 10;
187         if(tipcommand2.compareTo("√") == 0) Tipcode2 = 11;
188         if(tipcommand2.compareTo("^") == 0) Tipcode2 = 12;
189         //显示帮助和错误信息
190         TipShow(bracket , Tipcode1 , Tipcode2 , tipcommand1 ,
            tipcommand2);
191     }
```

3.3.5　TipShow()函数

显示Tip信息的函数是TipShow()，这个函数很简单，根据传入的错误代码或者帮助代码的值显示相应信息。

```
001 /*
002 * 反馈 Tip 信息，加强人机交互，与 TipChecker()配合使用
003 */
004 private void TipShow(int bracket , int tipcode1 , int tipcode2 ,
005         String tipcommand1 , String tipcommand2) {
006     String tipmessage = "";
007     if(tipcode1 != 0)
008     tip_lock = false;                       //表明输入有误
009     switch(tipcode1) {
010     case -1:
011         tipmessage = tipcommand2 + "  不能作为第一个算符\n";
012         break;
013     case 1:
014         tipmessage = tipcommand1 + "  后应输入：数字/(/./-/函数 \n";
015         break;
016     case 2:
017         tipmessage = tipcommand1 + "  后应输入：)/算符 \n";
018         break;
019     case 3:
020         tipmessage = tipcommand1 + "  后应输入：)/数字/算符 \n";
021         break;
022     case 4:
023         tipmessage = tipcommand1 + "  后应输入：)/./数字 /算符 \n";
024         break;
025     case 5:
026         tipmessage = tipcommand1 + "  后应输入：(/./数字/函数 \n";
027         break;
028     case 6:
029         tipmessage = tipcommand1 + "  后应输入：(/./数字 \n";
030         break;
031     case 7:
032         tipmessage = tipcommand1 + "  后应输入：(/./数字 \n";
033         break;
034     case 8:
035         tipmessage = "小数点重复\n";
036         break;
037     case 9:
038         tipmessage = "不能计算，缺少 "+ bracket +" 个 )";
039         break;
040     case 10:
041         tipmessage = "不需要  )";
042         break;
043     }
044     switch(tipcode2) {
045     case 1:
046         tipmessage = tipmessage + "[MC 用法：清除记忆 MEM]";
047         break;
048     case 2:
049         tipmessage = tipmessage + "[C 用法：归零]";
050         break;
```

```
051     case 3:
052         tipmessage = tipmessage + "[DRG 用法: 选择 DEG 或 RAD]";
053         break;
054     case 4:
055         tipmessage = tipmessage + "[Bksp 用法: 退格]";
056         break;
057     case 5:
058         tipmessage = tipmessage + "sin 函数用法示例: \n" +
059                 "DEG: sin30 = 0.5      RAD: sin1 = 0.84\n" +
060                 "注: 与其他函数一起使用时要加括号, 如: \n" +
061                 "sin(cos45), 而不是 sincos45" ;
062         break;
063     case 6:
064         tipmessage = tipmessage + "cos 函数用法示例: \n" +
065                 "DEG: cos60 = 0.5      RAD: cos1 = 0.54\n" +
066                 "注: 与其他函数一起使用时要加括号, 如: \n" +
067                 "cos(sin45), 而不是 cossin45" ;
068         break;
069     case 7:
070         tipmessage = tipmessage + "tan 函数用法示例: \n" +
071                 "DEG: tan45 = 1      RAD: tan1 = 1.55\n" +
072                 "注: 与其他函数一起使用时要加括号, 如: \n" +
073                 "tan(cos45), 而不是 tancos45" ;
074         break;
075     case 8:
076         tipmessage = tipmessage + "log 函数用法示例: \n" +
077                 "log10 = log(5+5) = 1\n" +
078                 "注: 与其他函数一起使用时要加括号, 如: \n" +
079                 "log(tan45), 而不是 logtan45" ;
080         break;
081     case 9:
082         tipmessage = tipmessage + "ln 函数用法示例: \n" +
083                 "ln10 = le(5+5) = 2.3   lne = 1\n" +
084                 "注: 与其他函数一起使用时要加括号, 如: \n" +
085                 "ln(tan45), 而不是 lntan45" ;
086         break;
087     case 10:
088         tipmessage = tipmessage + "n! 函数用法示例: \n" +
089                 "n!3 = n!(1+2) = 3×2×1 = 6\n" +
090                 "注: 与其他函数一起使用时要加括号, 如: \n" +
091                 "n!(log1000), 而不是 n!log1000" ;
092         break;
093     case 11:
094         tipmessage = tipmessage + "√ 用法示例: 开任意次根号\n" +
095                 "如: 27 开 3 次根为  27√3 = 3\n" +
096                 "注: 与其他函数一起使用时要加括号, 如: \n" +
097                 "(函数)√(函数) ,  (n!3)√(log100) = 2.45";
098         break;
099     case 12:
100         tipmessage = tipmessage + "^ 用法示例: 开任意次平方\n" +
101                 "如: 2 的 3 次方为  2^3 = 8\n" +
102                 "注: 与其他函数一起使用时要加括号, 如: \n" +
103                 "(函数)√(函数) ,  (n!3)^(log100) = 36";
104         break;
105     }
106     //将提示信息显示到 tip
```

```
107     tip.setText(tipmessage);
108 }
```

3.3.6　计算类 calc

最后来看一下整个计算器的核心——计算类calc，如图 3.2 所示，描述了整个计算的流程。计算的过程从分析输入的表达式开始，首先要清楚以怎样的流程去解析这个表达式，要将我们平时的思维转换一下。人的思维可以是跳跃的，而电脑只能一步步来。因此将我们思考问题的方法抽象成电脑能够理解的算法，这是很多时候我们需要做的事，本章编写计算器的核心其实也就是这个算法。

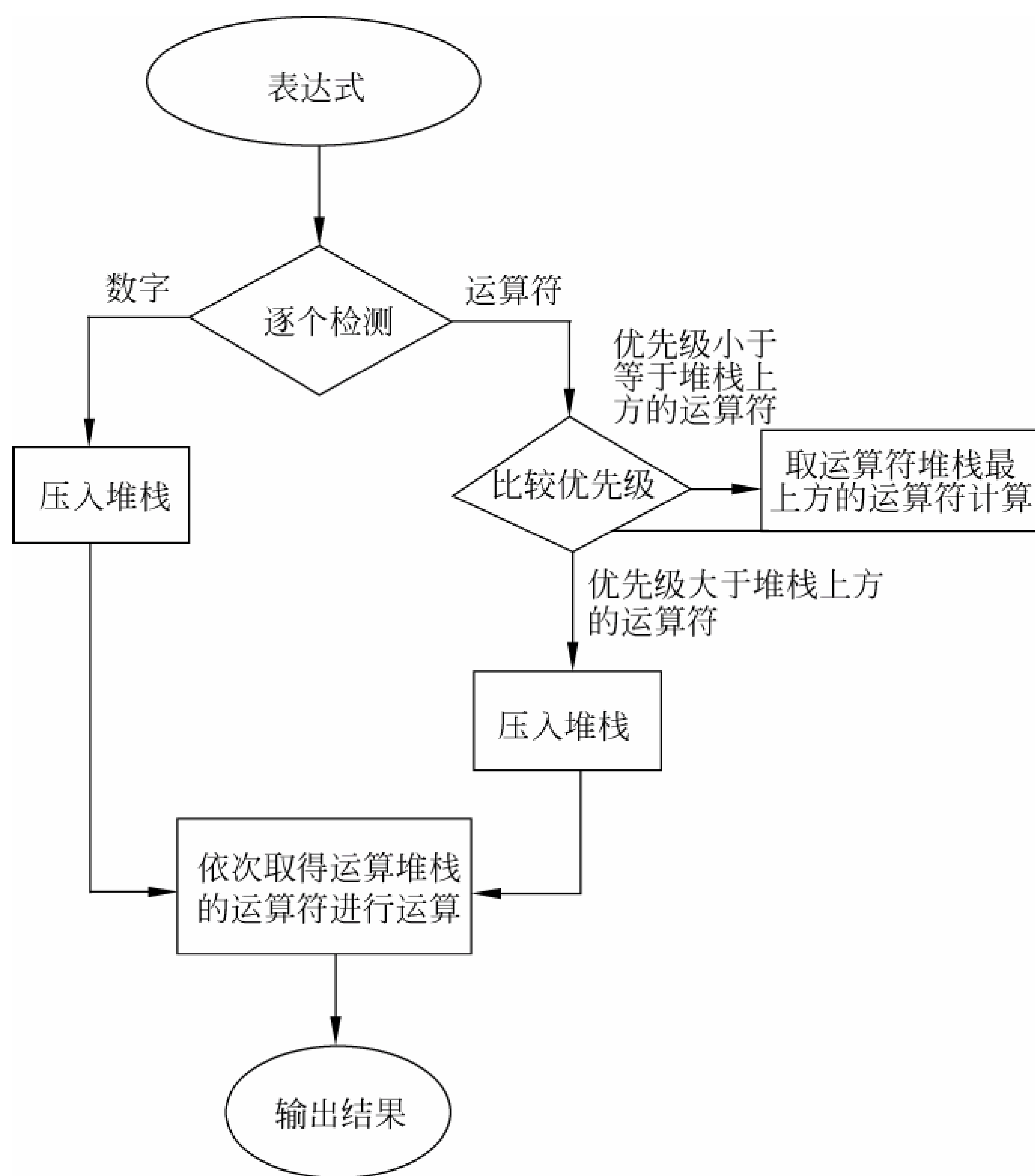


图 3.2　计算过程流程图

表达式主要包含两部分，一部分是数字，一部分是运算符号，首先要做的就是分离这两种元素。以下代码 024～035 行定义了一些变量，用于存储数字、运算符以及运算符的优先级。在代码 040～046 行将表达式中的负号提取出来，传递给flag，在 064 行中将flag的符号传递给数字，并将flag置为 1。通过这种方式，整个表达式的符号全部传递给数字，因此到最后运算的时候就不需要另外考虑结果的正负。

代码 048～066 行用于获取整个算式中的所有数字，并以此进入堆栈。前面分析过，这些数字的正负由它附近的正负标志位决定。

代码 068～100 行为运算符确定在算式中的优先级，运算符的优先级由两部分决定：基本优先级和变动优先级。基础优先级是固定不变的，比如+、-的优先级是一、×、÷的

优先级是二；而变化优先级由括号层数决定，一层括号的优先级加四。通过这样，我们就算出当前运算符的优先级。

接着要将运算符及对应的优先级别入栈。这时候需要作一个判断，如果当前运算符的优先级别高于堆栈顶端运算符的优先级，则直接入栈，如代码 102～105 行所示。否则，将堆栈顶部的运算符逐个取出进行计算，直到当前运算符堆栈顶部的运算符优先级别低于当前运算符，或者堆栈为空，如代码 107～219 行所示。这样，就能保证堆栈中的运算符优先级别是从高到低，便于之后进行计算。

运算符入了堆栈之后开始进行计算，如代码 221～323 行所示，依次从堆栈顶部取出运算符进行计算。因为前面我们已经保证了运算符的该堆栈中的优先级顺序，因此这时候只要依次取出计算就可以。

计算完之后，将计算结果进行有效位格式化，并显示到显示器上，整个计算过程结束。所有代码如下所示：

```
001    /*
002     * 整个计算核心，只要将表达式的整个字符串传入 calc().process()就可以实行计算了
003     * 算法包括以下几部分：
004     * 1. 计算部分      process(String str) 当然，这是建立在查错无错误的情况下
005     * 2. 数据格式化    FP(double n)        使数据有相当的精确度
006     * 3. 阶乘算法      N(double n)         计算 n!，将结果返回
007     * 4. 错误提示      showError(int code ,String str)  将错误返回
008     */
009    public class calc {
010        public calc(){
011
012       }
013        final int MAXLEN = 500;
014           /*
015            * 计算表达式
016            * 从左向右扫描，数字入 number 栈，运算符入 operator 栈
017            * +-基本优先级为 1，×÷基本优先级为 2，log ln sin cos tan n!基本优先级为 3，√^基本优先级为 4
018            * 括号内层运算符比外层同级运算符优先级高 4
019            * 当前运算符优先级高于栈顶压栈，低于栈顶弹出一个运算符与两个数进行运算
020            * 重复直到当前运算符大于栈顶
021            * 扫描完后对剩下的运算符与数字依次计算
022            */
023     public void process(String str) {
024         int weightPlus = 0, topOp = 0,
025         topNum = 0, flag = 1, weightTemp = 0;
026         //weightPlus 为同一（）下的基本优先级，weightTemp 临时记录优先级的变化
027         //topOp 为 weight[]、operator[]的计数器；topNum 为 number[]的计数器
028         //flag 为正负数的计数器，1 为正数，-1 为负数
029         int weight[];         //保存 operator 栈中运算符的优先级，以 topOp 计数
030         double number[];      //保存数字，以 topNum 计数
031         char ch, ch_gai, operator[];//operator[]保存运算符，以 topOp 计数
032         String num;//记录数字，str 以+-×÷()sctgl!√^分段，+-×÷()sctgl!√^字符之间的字符串即为数字
033         weight = new int[MAXLEN];
034         number = new double[MAXLEN];
035         operator = new char[MAXLEN];
```

```
036         String expression = str;
037         StringTokenizer expToken = new StringTokenizer(expression,
            "+-×÷()sctgl!√^");
038         int i = 0;
039         while (i < expression.length()) {
040             ch = expression.charAt(i);
041             //判断正负数
042             if (i == 0) {
043                 if (ch == '-')
044                     flag = -1;
045             } else if(expression.charAt(i-1) == '(' && ch == '-')
046                 flag = -1;
047             //取得数字，并将正负符号转移给数字
048             if (ch <= '9' && ch >= '0'|| ch == '.' || ch == 'E') {
049                 num = expToken.nextToken();
050                 ch_gai = ch;
051                 Log.e("guojs",ch+"--->"+i);
052                 //取得整个数字
053                 while (i < expression.length() &&
054                         (ch_gai <= '9' && ch_gai >= '0'|| ch_gai == '.' ||
                         ch_gai == 'E'))
055                 {
056                     ch_gai = expression.charAt(i++);
057                     Log.e("guojs","i 的值为: "+i);
058                 }
059                 //将指针退回之前的位置
060                 if (i >= expression.length()) i-=1; else {i-=2;}
061                 if (num.compareTo(".") == 0) number[topNum++] = 0;
062                 //将正负符号转移给数字
063                 else {
064                     number[topNum++] = Double.parseDouble(num)*flag;
065                     flag = 1;
066                 }
067             }
068             //计算运算符的优先级
069             if (ch == '(') weightPlus+=4;
070             if (ch == ')') weightPlus-=4;
071             if (ch == '-' && flag == 1 || ch == '+' || ch == '×'|| ch ==
            '÷' ||
072                     ch == 's' ||ch == 'c' || ch == 't' || ch == 'g' || ch
                    == 'l' ||
073                     ch == '!' || ch == '√' || ch == '^') {
074                 switch (ch) {
075                     //+-的优先级最低，为 1
076                     case '+':
077                     case '-':
078                         weightTemp = 1 + weightPlus;
079                         break;
080                     //x÷的优先级稍高，为 2
081                     case '×':
082                     case '÷':
083                         weightTemp = 2 + weightPlus;
084                         break;
085                     //sincos 之类的优先级为 3
086                     case 's':
087                     case 'c':
088                     case 't':
089                     case 'g':
090                     case 'l':
091                     case '!':
```

```
092                     weightTemp = 3 + weightPlus;
093                     break;
094                 //其余优先级为 4
095                 //case '^':
096                 //case '√':
097                 default:
098                     weightTemp = 4 + weightPlus;
099                     break;
100             }
101             //如果当前优先级大于堆栈顶部元素，则直接入栈
102             if (topOp == 0 || weight[topOp-1] < weightTemp) {
103                 weight[topOp] = weightTemp;
104                 operator[topOp] = ch;
105                 topOp++;
106             //否则将堆栈中的运算符逐个取出，直到当前堆栈顶部运算符的优先级小于当
                前运算符
107             }else {
108                  while (topOp > 0 && weight[topOp-1] >= weightTemp) {
109                     switch (operator[topOp-1]) {
110                         //取出数字数组的相应元素进行运算
111                         case '+':
112                             number[topNum-2]+=number[topNum-1];
113                             break;
114                         case '-':
115                             number[topNum-2]-=number[topNum-1];
116                             break;
117                         case '×':
118                             number[topNum-2]*=number[topNum-1];
119                             break;
120                         //判断除数为 0 的情况
121                         case '÷':
122                             if (number[topNum-1] == 0) {
123                                 showError(1,str old);
124                                 return;
125                             }
126                             number[topNum-2]/=number[topNum-1];
127                             break;
128                         case '√':
129                             if(number[topNum-1] == 0 || (number[top
                            Num-2] < 0 &&
130                                     number[topNum-1] % 2 == 0)) {
131                                 showError(2,str_old);
132                                 return;
133                             }
134                             number[topNum-2] =
135                                 Math.pow(number[topNum-2], 1/number
                                 [topNum-1]);
136                             break;
137                         case '^':
138                             number[topNum-2] =
139                                 Math.pow(number[topNum-2], number[top
                                Num-1]);
140                             break;
141                         //计算时进行角度弧度的判断及转换
142                         //sin
143                         case 's':
144                             if(drg_flag == true) {
145                                 number[topNum-1] = Math.sin((number[top
                                Num-1]/180)*pi);
146                             } else {
```

```
147                         number[topNum-1] = Math.sin(number[top
                            Num-1]);
148                     }
149                     topNum++;
150                     break;
151                 //cos
152                 case 'c':
153                     if(drg_flag == true) {
154                         number[topNum-1] = Math.cos((number[top
                            Num-1]/180)*pi);
155                     } else {
156                         number[topNum-1] = Math.cos(number[top
                            Num-1]);
157                     }
158                     topNum++;
159                     break;
160                 //tan
161                 case 't':
162                     if(drg_flag == true) {
163                         if((Math.abs(number[topNum-1])/90)%2
                            == 1) {
164                             showError(2,str_old);
165                             return;
166                         }
167                         number[topNum-1] = Math.tan((number[top
                            Num-1]/180)*pi);
168                     } else {
169                         if((Math.abs(number[topNum-1])/(pi/2))%2
                            == 1) {
170                             showError(2,str_old);
171                             return;
172                         }
173                         number[topNum-1] = Math.tan(number[top
                            Num-1]);
174                     }
175                     topNum++;
176                     break;
177                 //log
178                 case 'g':
179                     if(number[topNum-1] <= 0) {
180                         showError(2,str_old);
181                         return;
182                     }
183                     number[topNum-1] = Math.log10(number[top
                        Num-1]);
184                     topNum++;
185                     break;
186                 //ln
187                 case 'l':
188                     if(number[topNum-1] <= 0) {
189                         showError(2,str_old);
190                         return;
191                     }
192                     number[topNum-1] = Math.log(number[top
                        Num-1]);
193                     topNum++;
194                     break;
195                 //阶乘
196                 case '!':
197                     if(number[topNum-1] > 170) {
198                         showError(3,str_old);
```

```
199                             return;
200                         } else if(number[topNum-1] < 0) {
201                             showError(2,str old);
202                             return;
203                         }
204                         number[topNum-1] = N(number[topNum-1]);
205                         topNum++;
206                         break;
207                     }
208                     //继续取堆栈的下一个元素进行判断
209                     topNum--;
210                     topOp--;
211                 }
212                 //将运算符压入堆栈
213                 weight[topOp] = weightTemp;
214                 operator[topOp] = ch;
215                 topOp++;
216             }
217         }
218         i++;
219     }
220     //依次取出堆栈的运算符进行运算
221     while (topOp>0) {
222         //+-x 直接将数组的后两位数取出运算
223         switch (operator[topOp-1]) {
224         case '+':
225             number[topNum-2]+=number[topNum-1];
226             break;
227         case '-':
228             number[topNum-2]-=number[topNum-1];
229             break;
230         case '×':
231             number[topNum-2]*=number[topNum-1];
232             break;
233         //涉及到除法时要考虑除数不能为零的情况
234         case '÷':
235             if (number[topNum-1] == 0) {
236                 showError(1,str old);
237                 return;
238             }
239             number[topNum-2]/=number[topNum-1];
240             break;
241         case '√':
242             if(number[topNum-1] == 0 || (number[topNum-2] < 0 &&
243                     number[topNum-1] % 2 == 0)) {
244                 showError(2,str_old);
245                 return;
246             }
247             number[topNum-2] =
248                 Math.pow(number[topNum-2], 1/number[topNum-1]);
249             break;
250         case '^':
251             number[topNum-2] =
252                 Math.pow(number[topNum-2], number[topNum-1]);
253             break;
254         //sin
255         case 's':
256             if(drg flag == true) {
257                 number[topNum-1] = Math.sin((number[topNum-1]
                    /180)*pi);
```

```
258                 } else {
259                     number[topNum-1] = Math.sin(number[topNum-1]);
260                 }
261                 topNum++;
262                 break;
263             //cos
264             case 'c':
265                 if(drg_flag == true) {
266                     number[topNum-1] = Math.cos((number[topNum-1]/
                    180)*pi);
267                 } else {
268                     number[topNum-1] = Math.cos(number[topNum-1]);
269                 }
270                 topNum++;
271                 break;
272             //tan
273             case 't':
274                 if(drg_flag == true) {
275                     if((Math.abs(number[topNum-1])/90)%2 == 1) {
276                         showError(2,str_old);
277                         return;
278                     }
279                     number[topNum-1] = Math.tan((number[topNum-1]/
                    180)*pi);
280                 } else {
281                     if((Math.abs(number[topNum-1])/(pi/2))%2 == 1) {
282                         showError(2,str_old);
283                         return;
284                     }
285                     number[topNum-1] = Math.tan(number[topNum-1]);
286                 }
287                 topNum++;
288                 break;
289             //对数 log
290             case 'g':
291                 if(number[topNum-1] <= 0) {
292                     showError(2,str_old);
293                     return;
294                 }
295                 number[topNum-1] = Math.log10(number[topNum-1]);
296                 topNum++;
297                 break;
298             //自然对数 ln
299             case 'l':
300                 if(number[topNum-1] <= 0) {
301                     showError(2,str_old);
302                     return;
303                 }
304                 number[topNum-1] = Math.log(number[topNum-1]);
305                 topNum++;
306                 break;
307             //阶乘
308             case '!':
309                 if(number[topNum-1] > 170) {
310                     showError(3,str_old);
311                     return;
312                 } else if(number[topNum-1] < 0) {
313                     showError(2,str_old);
314                     return;
315                 }
316                 number[topNum-1] = N(number[topNum-1]);
```

```
317                 topNum++;
318                 break;
319             }
320             //取堆栈下一个元素计算
321             topNum--;
322             topOp--;
323         }
324         //如果数字太大，提示错误信息
325         if(number[0] > 7.3E306) {
326             showError(3,str old);
327             return;
328         }
329         //输出最终结果
330         input.setText(String.valueOf(FP(number[0])));
331         tip.setText("计算完毕，要继续请按归零键 C");
332         mem.setText(str old+"="+String.valueOf(FP(number[0])));
333     }
334
335     /*
336      * FP = floating point 控制小数位数，达到精度
337      * 否则会出现 0.6-0.2=0.39999999999999997 的情况，用 FP 即可解决，使得数为
         0.4
338      * 本格式精度为 15 位
339      */
340     public double FP(double n) {
341         //NumberFormat format=NumberFormat.getInstance();
                                                    //创建一个格式化类
342         //format.setMaximumFractionDigits(18);  //设置小数位的格式
343         DecimalFormat format = new DecimalFormat("0.##############");
344         return Double.parseDouble(format.format(n));
345     }
346
347     /*
348      * 阶乘算法
349      */
350     public double N(double n) {
351         int i = 0;
352         double sum = 1;
353         //依次将小于等于 n 的值相乘
354         for(i = 1;i <= n;i++) {
355             sum = sum*i;
356         }
357         return sum;
358     }
359
360     /*
361      * 错误提示，按了“=”之后，若计算式在 process()过程中出现错误，则进行提示
362      */
363     public void showError(int code ,String str) {
364         String message="";
365         switch (code) {
366         case 1:
367             message = "零不能作除数";
368             break;
369         case 2:
370             message = "函数格式错误";
371             break;
372         case 3:
```

```
373             message = "值太大了，超出范围 ";
374         }
375         input.setText("\""+str+"\""+": "+message);
376         tip.setText(message+"\n"+"计算完毕，要继续请按归零键 C");
377     }
378 }
```

最后来看看运行的效果图，如图 3.3 所示。



图 3.3　计算结果显示

3.4　知识拓展

在对计算表达式进行分析的时候我们用到了StringTokenizer这个类，接下来来看看StringTokenizer的用法：

```
StringTokenizer expToken = new StringTokenizer(expression,"+-×÷()sctgl!
√^");
```

StringTokenizer是一个用来分隔String的应用类，用法类似于split。StringTokenizer类的构造函数有 3 种：

- ❑ public StringTokenizer(String str)：不指定分隔字符，默认的分隔符是空格、制表符(\t)、换行符(\n)和回车符(\r)。
- ❑ public StringTokenizer(String str, String delim)：构造一个用来解析 str 的 StringTokenizer 对象，并提供一个指定的分隔符。
- ❑ public StringTokenizer(String str, String delim, boolean returnDelims)：构造一个用来解析 str 的 StringTokenizer 对象，并提供一个指定的分隔符，同时，指定是否返回分隔符。

StringTokenizer的核心函数有 4 个，如下所示：

- ❑ public boolean hasMoreTokens()：返回是否还有分隔符。

- public String nextToken()：返回从当前位置到下一个分隔符的字符串。
- public String nextToken(String delim)：返回从当前位置到下一个指定分隔符的字符串。
- public int countTokens()：返回 nextToken 方法被调用的次数。如果采用第一种或者第二种构造函数，返回的就是分隔符数量。

3.5　本 章 小 结

本章主要介绍了如何开发一个计算器的应用程序，采用TableRow进行布局设计，功能方面完全按照现实中的计算器进行设计。本章的重点在于计算过程的流程以及相应的算法，读者要充分认识到算法通常不同于我们平常的思维，而是采用一种流程的方式来进行思考。

第 4 章　电 子 词 典

电子词典是 Android 上很常用的应用，一般用户手机都会安装一个电子词典。电子词典的好坏有一半以上取决于词库，没有好的词库，其他的都是摆设。除了要有好的词库，友好的用户界面和便捷的使用方式以及快速的反应能力都极大地影响着用户对电子词典的使用感受。本章将要开发一款简单的电子词典，功能很简单，就是将英文翻译成中文。

4.1　功 能 分 析

对于电子词典的核心部件——词库，我们有两种获取方式，一种是本地词库，一般是以数据库的方式存在；另一种是网络词库，也就是通过解析类似于 Google 翻译这种网站的信息，将用户的查询信息输入到在线翻译的网站，然后提取出结果。可以看出这两种方式的一个最大的区别就在于，一个是本地一个是在线。本章将要开发的电子词典是基于本地词库的，这种方式的好处在于，不受限于网络，查询速度快，当然缺点很明显，就是有可能出现词库容量有限导致的某些单词查询不到或者解释不够全面的情况。

电子词典设计运行的效果如图 4.1 所示，顶部是一个 AutoCompleteTextView 组件，当用户在组件中输入两个或两个以上字母的时候，后台会开始进行数据库查询，将以这两个字母开头的单词显示到一个下拉列表中，供用户参考。用户可以从下拉列表的单词中选择，最后单击“搜索”，单词的中文意思将会显示到最下面的 TextView 中。

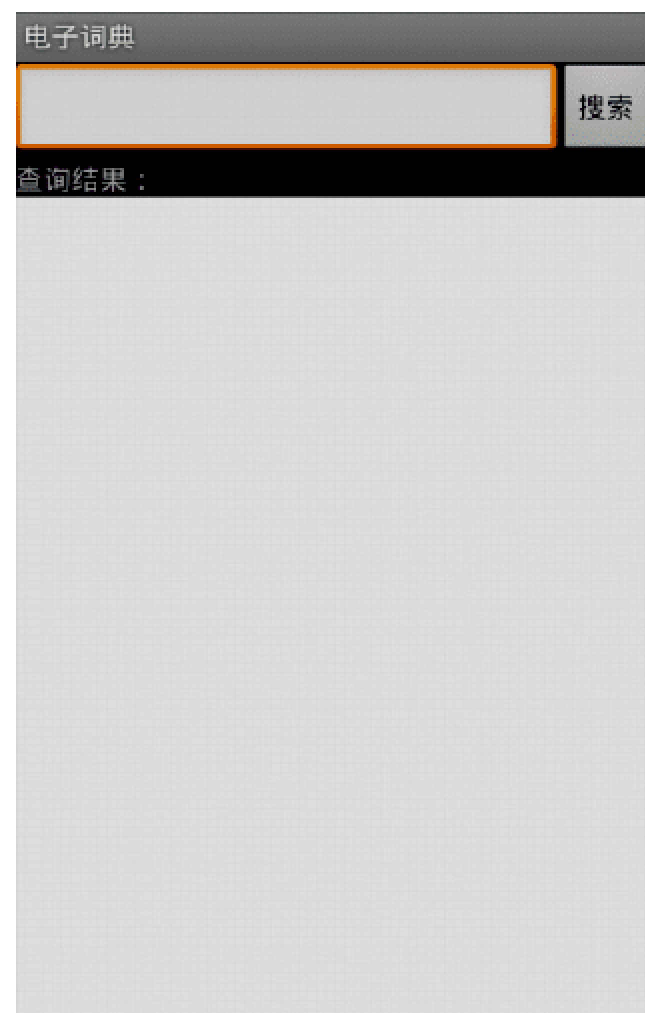


图 4.1　电子词典界面

因此除了后台进行数据库查询动作外，我们要做的就是将查询结果匹配到这个 AutoCompleteTextView 中。下面我们用一个小例子来看一下 AutoCompleteTextView 是如何使用的。

首先我们在 Eclipse 中新建一个工程 AutoCompleteTextView，在主界面程序文件 AutoCompleteDemo.java 中为组件 AutoCompleteTextView 绑定数组监听器，如下所示：

```
01  package com.supermario.autocompletedemo;                //声明包语句
02~05 行为引入相关类，这里不再列举，请阅读光盘内容
//……
06  public class AutoCompleteDemo extends Activity {
07      //创建一个字符串数组，用于为 adapter 提供数据来源
```

```
08      String GAME[]=new String[]{"game1","gam2","gamm3","gamek3"};
09      @Override
10      public void onCreate(Bundle savedInstanceState) {
11          super.onCreate(savedInstanceState);
12          setContentView(R.layout.main);
13         AutoCompleteTextView auto=(AutoCompleteTextView)findViewById
           (R.id.autoCompleteTextView1);
14          //吸纳进数组适配器 adapter，绑定数据，设定界面为 R.layout.list_view
15          ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,R.
            layout.list_view,GAME);
16          //为 AutoCompleteTextView 设定适配器
17          auto.setAdapter(adapter);
18      }
19  }
```

这里的下拉列表布局是我们在 layout 目录下自定义的，只需要定义单独的 TextView 即可，不需要其他的元素，如下所示：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <TextView
03      xmlns:android="http://schemas.android.com/apk/res/android"
04      android:id="@+id/textView1"
05      android:layout_width="match_parent"
06      android:layout_height="wrap_content"
07      android:gravity="center_vertical"
08      android:minHeight="?android:attr/listPreferredItemHeight"
09      android:textColor="#999999" />
```

而主界面 main.xml 中则只有一个 AutoCompleteTextView，如下所示，其中的 <requestFocus />标签用于指定屏幕内的焦点 View。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
03      android:layout_width="fill_parent"
04      android:layout_height="fill_parent"
05      android:orientation="vertical" >
06      <!-- 自动完成文本框-->
07      <AutoCompleteTextView
08          android:id="@+id/autoCompleteTextView1"
09          android:layout_width="match_parent"
10          android:layout_height="wrap_content"
11          android:ems="10"
12          android:hint="请选择你喜欢的游戏"
13          android:completionHint="请选择你喜欢的游戏"
14          android:singleLine="true" >
15          <requestFocus />
16      </AutoCompleteTextView>
17  </LinearLayout>
```

最后我们在模拟器中运行该程序，在文本框中输入“ga”，则弹出下拉列表供我们选择，如图 4.2 所示。



图 4.2　AutoCompleteTextView 示例

4.2　界面设计

整个界面设计很简洁，代码如下所示，主要提供了一个供用户输入的文本框和一个查询按钮以及一个显示查询结果的 TextView。在处理一排放置两个元素的时候，我们加入了 RelativeLayout 用于并排显示两个元素。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout width="fill parent"
04      android:layout height="fill parent"
05      android:orientation="vertical" >
06      <!-- 使用 RelativeLayout 布局用于在一行中显示两个元素 -->
07      <RelativeLayout
08          android:id="@+id/search"
09          android:layout width="match parent"
10          android:layout height="wrap content" >
11          <!-- 定义自动完成文本框，供用户输入单词 -->
12          <AutoCompleteTextView
13              android:id="@+id/word"
14              android:layout_width="fill_parent"
15              android:layout_height="wrap_content"
16              android:layout_alignParentLeft="true"
17              android:layout_toLeftOf="@+id/searchWord"
18              android:ems="20"
19              android:singleLine="true" >
20              <requestFocus />
21          </AutoCompleteTextView>
22          <!-- 搜索按钮，供用户单击查询单词 -->
23          <Button
24              android:id="@+id/searchWord"
```

```
25              android:layout_width="wrap_content"
26              android:layout_height="wrap_content"
27              android:layout_alignParentRight="true"
28              android:layout_alignParentTop="true"
29              android:text="@string/searchWord" />
30      </RelativeLayout>
31      <!-- 查询结果 -->
32      <TextView
33          android:id="@+id/textView1"
34          android:layout_width="wrap_content"
35          android:layout_height="wrap_content"
36          android:text="@string/searchLable" />
37      <!-- 用户显示查询的结果 -->
38      <TextView
39          android:id="@+id/result"
40          android:layout_width="match_parent"
41          android:layout_height="wrap_content"
42          android:layout_weight="0.68"
43          android:background="@color/white"
44          android:textColor="@color/blue" />
45  </LinearLayout>
```

4.3　功 能 实 现

整个应用程序的功能实现都包含在 Dictionary.java 这个文件里面，逻辑也相对简单。

（1）首先，我们先定义几个需要用到的变量，如下所示：

```
01  //定义数据库的存放路径
02  private final String DATABASE_PATH = android.os.Environment
03          .getExternalStorageDirectory().getAbsolutePath()
04          + "/dictionary";
05  //用户输入文本框
06  private AutoCompleteTextView word;
07  //定义数据库的名字
08  private final String DATABASE_FILENAME = "dictionary.db";
09  private SQLiteDatabase database;
10  //搜索按钮
11  private Button searchWord;
12  //用户显示查询结果
13  private TextView showResult;
```

（2）接着在程序的初始化函数中初始化这些界面元素变量，打开数据库，以备查询使用，同时为按键绑定监听器，为文本框绑定文本改变监听器，如下所示：

```
01  @Override
02  public void onCreate(Bundle savedInstanceState)
03  {
04      super.onCreate(savedInstanceState);
05      setContentView(R.layout.main);
06      //打开数据库
07      database = openDatabase();
08      searchWord = (Button) findViewById(R.id.searchWord);
09      word = (AutoCompleteTextView) findViewById(R.id.word);
10      //绑定监听器
11      searchWord.setOnClickListener(this);
```

```
12      //绑定文字改变监听器
13      word.addTextChangedListener(this);
14      showResult=(TextView)findViewById(R.id.result);
15  }
```

（3）接下去我们来看一下打开数据库的函数 openDatabase()，如下所示。由于我们将数据库存放在 SD 目录中，因此我们首先要判断该数据库是否存在，如果不存在则将工程中的数据库文件复制过去，如代码 05～30 行所示。接着用 OpenOrCreateDatabase()函数打开数据库，并返回，如果这过程中出错了就返回一个 null。

```
01  private SQLiteDatabase openDatabase()
02  {
03      try
04      {
05          // 获得 dictionary.db 文件的绝对路径
06          String databaseFilename = DATABASE_PATH + "/" + DATABASE_
            FILENAME;
07          File dir = new File(DATABASE_PATH);
08          // 如果/sdcard/dictionary 目录不存在，则创建这个目录
09          if (!dir.exists())
10              dir.mkdir();
11          // 如果在/sdcard/dictionary 目录中不存在
12          // dictionary.db 文件，则从 res/raw 目录中复制这个文件到
13          // SD 卡的目录（/sdcard/dictionary）
14          if (!(new File(databaseFilename)).exists())
15          {
16              // 获得封装 dictionary.db 文件的 InputStream 对象
17              InputStream is = getResources().openRawResource(
18                      R.raw.dictionary);
19              FileOutputStream fos = new FileOutputStream(database
                Filename);
20              byte[] buffer = new byte[8192];
21              int count = 0;
22              // 开始复制 dictionary.db 文件
23              while ((count = is.read(buffer)) > 0)
24              {
25                  fos.write(buffer, 0, count);
26              }
27              //关闭文件流
28              fos.close();
29              is.close();
30          }
31          // 打开/sdcard/dictionary 目录中的 dictionary.db 文件
32          SQLiteDatabase database = SQLiteDatabase.openOrCreateDatabase(
33                  databaseFilename, null);
34          return database;
35      }
36      catch (Exception e)
37      {
38      }
39      //如果打开出错，则返回 null
40      return null;
41  }
```

OpenOrCreateDatabase() 这个函数有两个版本，一个是 Context.OpenOrCreate Database()，一个是 SQLiteDatabase. OpenOrCreateDatabase()。它们本质上完成的功能都一样，Context.openOrCreateDatabase 最终是需要调用 SQLite Database.openOrCreateDatabase

来完成数据库的创建的。也就是说，SQLiteDatabase 类是 Android 对 sqlite 的最底层的封装，几乎所有的对数据库的操作最终都通过这个类来实现。而 Context 里面提供的方法，是用于上下文的时候创建数据库，例如你在某个逻辑里面创建的数据库只是在特定的 Context 里面，对于数据库的权限，交由 Context 来管理，而这个逻辑可能会提供给不止一个 Context。

（4）在按键事件监听器中，我们根据当前单词输入文本框中的内容进行查询，如果查找到数据则将数据显示出来，否则显示“未找到该单词”。这里有两个地方需要注意，一个是查询完成之后需要用 cursor.moveToFirst()将游标的位置放在查询结果的第一个位置，因为使用 rawQuery 返回的结果中游标的位置在第一个记录之前。另一个是要对查询结果做一些字符串的处理，如代码 13 行所示，因为数据库在存入字符的时候会对一些特殊字符进行转换，因此显示的时候需要再转换回来。

```
01  public void onClick(View view)
02  {
03      //查询指定的单词
04      String sql = "select chinese from t words where english=?";
05      Cursor cursor = database.rawQuery(sql, new String[]
06      {word.getText().toString()});
07          String result = "未找到该单词.";
08          //  如果查找到单词，显示其中文的意思
09          if (cursor.getCount() > 0)
10          {
11              //  必须使用 moveToFirst 方法将记录指针移动到第 1 条记录的位置
12              cursor.moveToFirst();
13              result = cursor.getString(cursor.getColumnIndex("chinese"))
                .replace("&amp;", "&");
14          }
15          //将结果显示到 TextView 中
16          showResult.setText(word.getText()+"\n"+result.toString());
17  }
```

（5）本程序中的 AutoCompleteTextView 绑定的 Adapter 跟之前有所不同，之前的 Adapter 是绑定一个常数数组，每次输入两个以上字符的时候就去匹配这个数组。但是，我们不可能把整个词库的单词全部做成一个常数数组再让组件自动去匹配，可以想象这样效率有多低。因此我们采用的做法是在 AutoCompleteTextView 类的 afterTextChanged 事件中监视 AutoCompleteTextView 组件中字符的输入情况，每当输入一个字符时就生成一个 DictionaryAdapter 对象，然后将新生成的 DictionaryAdapter 对象与 AutoCompleteTextView 关联。显示以输入字符串开头的单词列表的效果如图 4.3 所示。

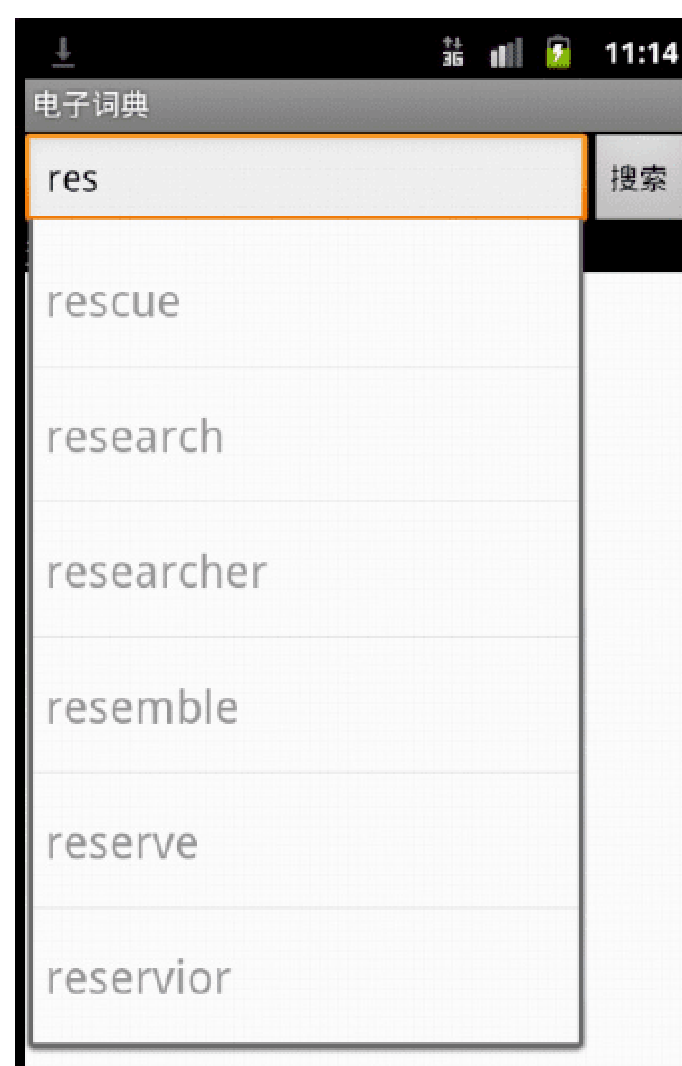


图 4.3　自动匹配相似的单词

在编写 DictionaryAdapter 类时应注意如下 3 点：

- 为了将 Cursor 对象与 AutoCompleteTextView 组件绑定，DictionaryAdapter 类必须从 CursorAdapter 类继承。
- 由于 CursorAdapter 类中的 convertToString 方法直接返回了 Cursor 对象的地址，因此，在 DictionaryAdapter 类中必须覆盖 convertToString 方法，以返回当前选中

的单词。这里要注意一下，当选中 AutoCompleteTextView 组件中单词列表中某一个单词后，系统会用 convertToString 方法的返回值来设置 AutoCompleteTextView 组件中的文本。因此，必须使用 Cursor 的 getString 来获得相应的字段值。

- 由于将 Cursor 对象与 Adapter 绑定时必须要有一个叫“_id”的字段，因此，在本例中将 english 字段名映射成了“_id”字段。

讲到这里我们应该了解一下 dictionary.db 中的 t_words 表的结果，该表只有两个字段：english 和 chinese。分别表示单词的英文和中文描述。如果要获得单词的中文描述，只需要查找 chinese 字段即可。

```
01  //自定义 Adapter 类
02  public class DictionaryAdapter extends CursorAdapter
03  {
04      private LayoutInflater layoutInflater;
05      @Override
06      public CharSequence convertToString(Cursor cursor)
07      {
08          return cursor == null ? "" : cursor.getString(cursor
09                  .getColumnIndex("_id"));
10      }
11      //将单词信息显示到列表中
12      private void setView(View view, Cursor cursor)
13      {
14          TextView tvWordItem = (TextView) view;
15          tvWordItem.setText(cursor.getString(cursor.getColumnIndex
             ("_id")));
16      }
17      //绑定选项到列表中
18      @Override
19      public void bindView(View view, Context context, Cursor cursor)
20      {
21          setView(view, cursor);
22      }
23      //生成新的选项
24      @Override
25      public View newView(Context context, Cursor cursor, ViewGroup parent)
26      {
27          View view = layoutInflater.inflate(R.layout.word_list_item,
            null);
28          setView(view, cursor);
29          return view;
30      }
31      public DictionaryAdapter(Context context, Cursor c, boolean auto
        Requery)
32      {
33          super(context, c, autoRequery);
34          layoutInflater = (LayoutInflater) context
35                  .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
36      }
37  }
38  public void afterTextChanged(Editable s)
39  {
40        //必须将 english 字段的别名设为_id
41      Cursor cursor = database.rawQuery(
42            "select english as _id from t_words where english like ?",
43            new String[]
```

```
44              { s.toString() + "%" });
45      //新建的 Adapter
46      DictionaryAdapter dictionaryAdapter = new DictionaryAdapter(this,
47              cursor, true);
48      //绑定适配器
49      word.setAdapter(dictionaryAdapter);
50
51  }
```

（6）在 DictionaryAdapter 类中需要使用 bindView 和 newView 方法设置每一个列表项。bindView 方法负责设置已经存在的列表项，也就是该列表项已经生成了相应的组件对象。而 newView 方法负责设置新的列表项，在该方法中需要创建一个 View 对象来显示当前的列表项。在本例中使用 word_list_item.xml 布局文件来显示每一个列表项，代码如下：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <TextView xmlns:android="http://schemas.android.com/apk/res/android"
03      android:id="@+id/tvWordItem"
04      android:layout_width="fill_parent"
05      android:layout_height="wrap_content"
06      android:gravity="center_vertical"
07      android:minHeight="?android:attr/listPreferredItemHeight"
08      android:paddingLeft="6dip"
09      android:textAppearance="?android:attr/textAppearanceLarge"
10      android:textColor="@color/gray" />
```

（7）另外，从前面可以看到我们将数据库文件放在 res/raw 下面，如图 4.4 所示，放在这里的文件不会被压缩成二进制文件，而且可以通过 R 类来访问。除此之外，我们也可以将文件存放在 assets 目录中。这个目录中的文件除了不会被编译成二进制形式之外，另外一点就是，访问方式是通过文件名，而不是资源 ID。并且还有更重要的一点就是，大家可以在这里任意地建立子目录，而/res 目录中的资源文件是不能自行建立子目录的。

（8）最后我们来看一下运行效果图，如图 4.5 所示。

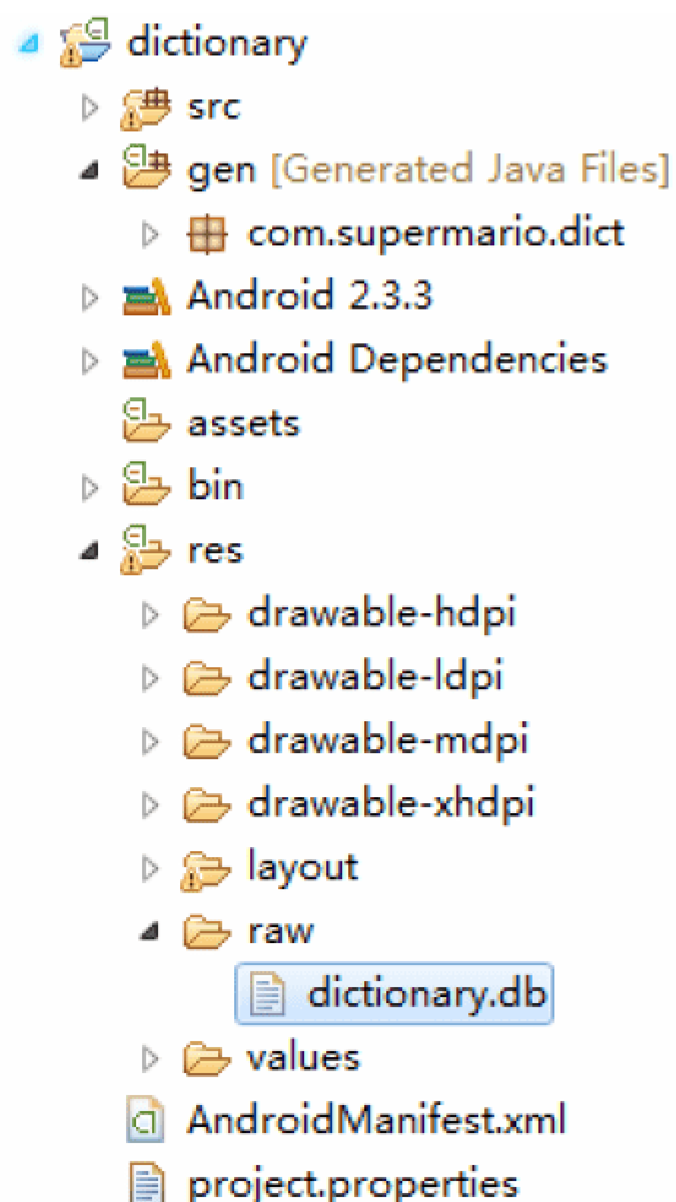


图 4.4　数据库文件存放位置



图 4.5　电子词典运行效果图

4.4 知 识 拓 展

本章在设计界面布局的时候用到了 RelativeLayout，RelativeLayout 顾名思义，相对布局。在这个容器内部的子元素们可以使用彼此之间的相对位置或者和容器间的相对位置来进行定位。在使用过程需要注意，不能在 RelativeLayout 容器本身和它的子元素之间产生循环依赖，比如说，不能将 RelativeLayout 的高设置成为 WRAP_CONTENT 的时候，将子元素的高设置成为 ALIGN_PARENT_BOTTOM。这点其实也是合理的规定，如果容器是不定高的，那么子元素当然无法对齐容器的底边。

RelativeLayout 里面的元素会用到一些特有的属性，这些属性主要用来定位，如下所示。

```
01  // 相对于给定 ID 控件
02  android:layout_above              //将该控件的底部置于给定 ID 的控件之上
03  android:layout_below              //将该控件的底部置于给定 ID 的控件之下
04  android:layout_toLeftOf           //将该控件的右边缘与给定 ID 的控件左边缘对齐
05  android:layout_toRightOf          //将该控件的左边缘与给定 ID 的控件右边缘对齐
06
07  android:layout_alignBaseline//将该控件的 baseline 与给定 ID 的 baseline 对齐
08  android:layout_alignTop           //将该控件的顶部边缘与给定 ID 的顶部边缘对齐
09  android:layout_alignBottom        //将该控件的底部边缘与给定 ID 的底部边缘对齐
10  android:layout_alignLeft          //将该控件的左边缘与给定 ID 的左边缘对齐
11  android:layout_alignRight         //将该控件的右边缘与给定 ID 的右边缘对齐
12  // 相对于父组件
13  android:layout_alignParentTop//如果为 true，将该控件的顶部与其父控件的顶部对齐
14  android:layout_alignParentBottom
                         //如果为 true，将该控件的底部与其父控件的底部对齐
15  android:layout_alignParentLeft//如果为 true，将该控件的左部与其父控件的左部对齐
16  android:layout_alignParentRight//如果为 true，将该控件的右部与其父控件的右部对齐
17  // 居中
18  android:layout_centerHorizontal//如果为 true，将该控件置于水平居中
19  android:layout_centerVertical     //如果为 true，将该控件置于垂直居中
20  android:layout_centerInParent     //如果为 true，将该控件置于父控件的中央
21  // 指定移动像素
22  android:layout_marginTop          //上偏移的值
23  android:layout_marginBottom       //下偏移的值
24  android:layout_marginLeft         //左偏移的值
25  android:layout_marginRight        //右偏移的值
26
27  example:
28  android:layout_below = "@id/***"
29  android:layout_alignBaseline = "@id/***"
30  android:layout_alignParentTop = true
31  android:layout_marginLeft = “10px”
```

接下去我们来看一个例子，代码如下所示，可以实现一个图片在左边，右面是一个标题，标题下是一行描述的典型布局。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout
03      xmlns:android="http://schemas.android.com/apk/res/android"
```

```
04      android:layout width="fill parent"
05      android:layout height="fill parent"
06   >
07   <!--位于左边的图像  -->
08   <ImageView
09       android:id="@+id/channellogo"
10       android:layout_height="wrap_content"
11       android:src="@drawable/ic_launcher"
12       android:layout_width="wrap_content">
13   </ImageView>
14  <!-- 位于右上方的文本 -->
15   <TextView
16       android:id="@+id/startime"
17       android:layout width="wrap content"
18       android:textSize="16dp"
19       android:layout height="wrap content"
20       android:padding="3dp"
21       android:scrollbars="vertical"
22       android:textColorHighlight="#ff0000ff"
23       android:text="title"
24       android:layout_toRightOf="@+id/channellogo" />
25   <!-- 位于右下方的文本 -->
26   <TextView
27       android:id="@+id/program"
28       android:layout width="wrap content"
29       android:textSize="16dp"
30       android:layout height="wrap content"
31       android:padding="3dp"
32       android:scrollbars="vertical"
33       android:textColorHighlight="#ffff0000"
34       android:layout toRightOf="@+id/channellogo"
35       android:text="description"
36       android:layout below="@+id/startime"/>
37  </RelativeLayout>
```

4.5　本 章 小 结

本章主要讲解通过本地数据库词库实现电子词典。本章所实现的电子词典功能比较简单，但却是一个电子词典的核心。本章技术的关键在于实现一个 CursorAdapter，通过这个适配器不断匹配用户当前输入的字符串，在后台查询，并把查询到的类似的字符串以列表方式显示出来，供用户选择。

另外本章还用到了 RelativeLayout，这种布局方式在后面我们会用得越来越多，熟练使用 RelativeLayout 这种布局方式，可以设计出复杂而错落有致的布局。

第 5 章　文件管理器

Android 系统并不自带文件管理器，但是很多情况下，我们有诸如从 SD 中打开文件的需要，怎么办呢？相信大家都比较习惯 Window 下操作文件和文件夹的方式，那么 Android 下是否也有类似的工具呢？答案是必须有。本章我们将要开发的应用就是 Android 平台下的文件管理器。

5.1　功 能 分 析

本章将要实现的文件管理器，借鉴于 Windows，从用户实际使用需求出发，主要实现的功能有：浏览任意目录下的文件及文件夹、打开文件、新建文件、删除文件、复制文件、对文件进行重命名、在当前目录或者整个目录进行搜索、返回上一级及根目录等。我们设计的文件管理器最终效果如图 5.1 所示。

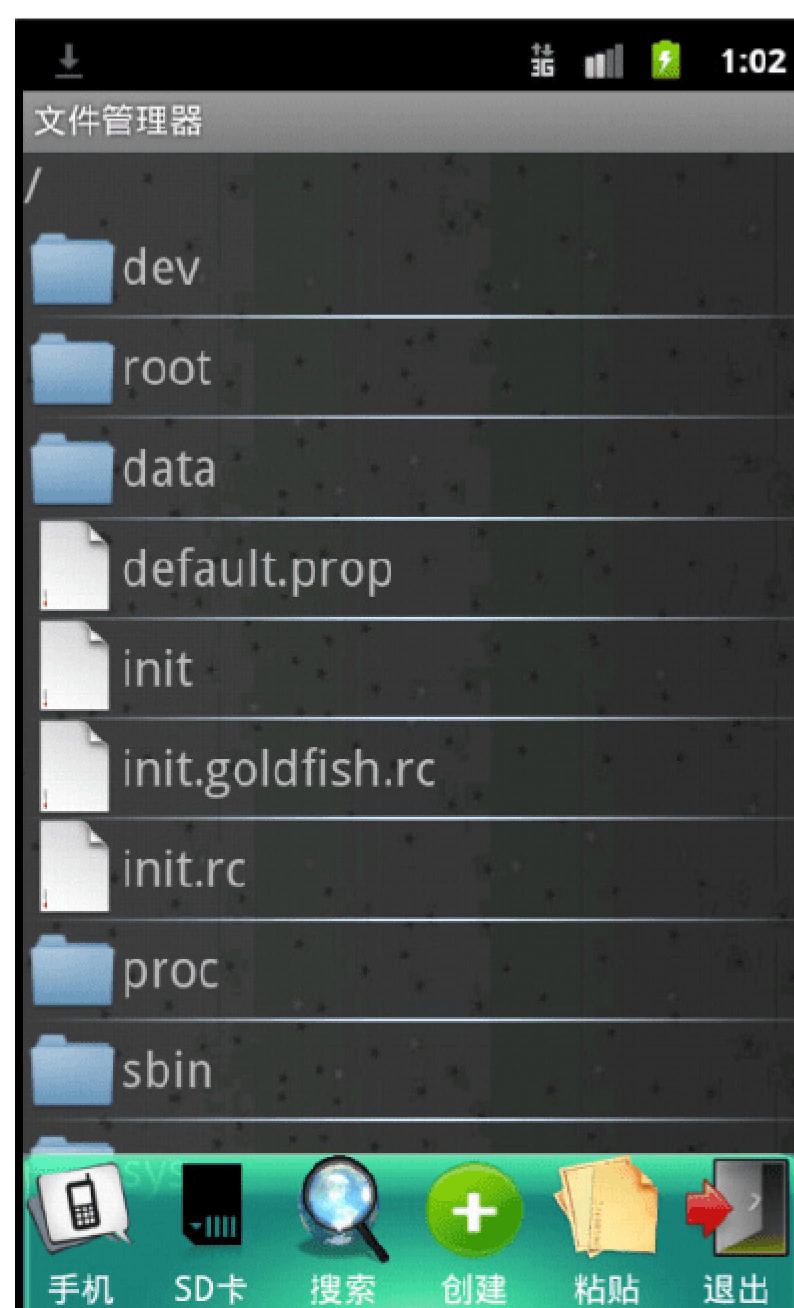


图 5.1　文件管理器

从图 5.1 可以看到，我们需要两种基本的布局：ListView 和 GridView。ListView 用于显示文件图标及图片，而 GridView 用于显示底部的菜单。并且，我们要为每一个元素绑定监听器，包括菜单项的单击监听器和列表菜单的长按监听器。同时我们可以想到，我们需

要用到多种形式的对话框，包括搜索对话框、重命名对话框等。

通过以上的分析，我们大体可以知道我们需要定制一些什么样的界面了。

5.2　界 面 设 计

（1）首先是主界面 main.xml，如上面分析，我们采用 RelativeLayout 的布局，在界面顶部放置一个 TextView 用于显示当前文件的路径，在底部放一个 GridView 用于放置菜单，而中间用一个 ListView 显示文件信息。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout
03      xmlns:android="http://schemas.android.com/apk/res/android"
04      android:layout_width="fill_parent"
05      android:layout_height="fill_parent"
06      android:background="@drawable/background">
07      <!-- 用于显示当前路径 -->
08      <TextView
09          android:layout width="fill parent"
10          android:layout height="wrap content"
11          android:id="@+id/mPath"
12          android:textSize="20sp"
13          android:singleLine="true"></TextView>
14      <!-- 用于显示当前目录的内容 -->
15      <ListView
16          android:id="@+id/android:list"
17          android:layout width="fill parent"
18          android:layout height="wrap content"
19          android:layout below="@+id/mPath"
20          android:cacheColorHint="#00000000"
21          android:divider="@drawable/line"
22          android:layout marginBottom="70px"></ListView>
23      <!-- 用于显示菜单 -->
24      <GridView
25          android:id="@+id/file_gridview_toolbar"
26          android:layout_height="wrap_content"
27          android:layout_width="fill_parent"
28          android:layout_alignParentBottom="true"></GridView>
29  </RelativeLayout>
```

（2）如下所示是创建文件时显示的对话框 create_dialog.xml，提供用户一个二选一选项，以及一个用于填写文件名称的文本框。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout
03    xmlns:android="http://schemas.android.com/apk/res/android"
04    android:layout width="fill parent"
05    android:layout height="wrap content"
06    android:orientation="vertical" >
07    <!-- 二选一 -->
08  <RadioGroup
09      android:id="@+id/radiogroup_create"
10      android:layout_width="fill_parent"
11      android:layout_height="wrap_content">
12      <!-- 创建文件选项 -->
```

```
13      <RadioButton
14      android:layout height="wrap content"
15      android:layout width="fill parent"
16      android:text="@string/create file"
17      android:id="@+id/create file" />
18      <!-- 创建文件夹选项 -->
19      <RadioButton
20      android:layout_height="wrap_content"
21      android:layout_width="fill_parent"
22      android:text="@string/create_folder"
23      android:id="@+id/create_folder" />
24  </RadioGroup>
25      <!-- 文本框，供用户填写文件名称 -->
26      <EditText
27      android:layout height="wrap content"
28      android:id="@+id/new filename"
29      android:layout width="fill parent"
30      android:hint="@string/create  hint"
31      android:singleLine="true" />
32  </LinearLayout>
```

（3）以下是重命名的对话框 rename_dialog.xml，只需要一个填写新文件名称的 EditText 即可。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout
03    xmlns:android="http://schemas.android.com/apk/res/android"
04    android:layout_height="wrap_content" android:layout_width="fill_
    parent">
05    <!-- 用于填写新的文件名称 -->
06    <EditText android:id="@+id/new filename"
07                    android:layout width="fill parent"
08                    android:layout height="wrap content"
09                    android:autoText="false"
10                    android:singleLine="true"
11                    android:capitalize="none"
12                    android:gravity="fill_horizontal"/>
13  </LinearLayout>
```

（4）接下去是搜索对话框 search_dialog.xml，同样提供一个二选一的选项——“在当前目录下搜索”和“在整个目录中搜索”，以及一个 EditText 用于用户输入要搜索的关键词。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout
03    xmlns:android="http://schemas.android.com/apk/res/android"
04    android:layout width="fill parent"
05    android:orientation="vertical" android:layout height="wrap
    content">
06    <!-- 提供二选一选项 -->
07  <RadioGroup
08      android:id="@+id/radiogroup_search"
09      android:layout_width="fill_parent"
10      android:layout height="wrap content">
11      <!-- 在当前目录下搜索 -->
12      <RadioButton
13      android:layout height="wrap content"
14      android:layout width="fill parent"
15      android:text="@string/radio_currentpath"
```

```
16      android:id="@+id/radio currentpath" />
17      <!-- 在整个目录中搜索 -->
18      <RadioButton
19      android:layout_height="wrap_content"
20      android:layout_width="fill_parent"
21      android:text="@string/radio_wholepath"
22      android:id="@+id/radio_wholepath" />
23  </RadioGroup>
24      <!--输入文件或者文件夹名字  -->
25      <EditText
26      android:layout height="wrap content"
27      android:id="@+id/edit search"
28      android:layout width="fill parent"
29      android:hint="@string/edit search hint" android:singleLine="true" />
30  </LinearLayout>
```

（5）还有 GirdView 的每个 Item 的布局 item_menu.xml。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
03      android:id="@+id/RelativeLayout Item"
04      android:layout width="fill parent"
05      android:layout height="wrap content"
06      android:paddingBottom="5dip">
07      <!-- 用于显示菜单的图片 -->
08      <ImageView
09          android:id="@+id/item image"
10          android:layout centerHorizontal="true"
11          android:layout width="wrap content"
12          android:layout height="45dp"></ImageView>
13      <!-- 用于显示菜单的文字 -->
14      <TextView
15          android:layout_below="@id/item_image"
16          android:id="@+id/item_text"
17          android:layout_centerHorizontal="true"
18          android:layout_width="wrap_content"
19          android:layout_height="wrap_content"
20          android:textColor="#FFFFFFFF"></TextView>
21  </RelativeLayout>
```

（6）用于显示 ListView 各个子元素的布局跟 item_menu.xml 很相似，如下所示，也是由一个 ImageView 和一个 TextView 组成。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:id="@+id/RelativeLayout Item"
04      android:layout width="fill parent"
05      android:layout height="wrap content"
06      android:paddingBottom="5dip">
07      <!-- 用于显示菜单的图片 -->
08      <ImageView
09          android:id="@+id/item_image"
10          android:layout_centerHorizontal="true"
11          android:layout_width="wrap_content"
12          android:layout_height="45dp"></ImageView>
13      <!-- 用于显示菜单的文字 -->
14      <TextView
15          android:layout_below="@id/item_image"
```

```
16          android:id="@+id/item_text"
17          android:layout_centerHorizontal="true"
18          android:layout_width="wrap_content"
19          android:layout_height="wrap_content"
```

（7）当打开文本文件的时候，我们会使用自己设计的一个编辑器，编辑器的布局文件 edit_txt.xml 如下所示，一个 TextView 用于显示文件的标题，一个 EditText 用于显示当前文件的内容，以及一个“保存”按钮和一个“取消”按钮。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout
03    xmlns:android="http://schemas.android.com/apk/res/android"
04    android:layout_width="fill_parent"
05    android:layout_height="fill_parent">
06  <!-- 用于显示文件名 -->
07  <TextView
08     android:id="@+id/TextViewTitle"
09     android:singleLine="true"
10     android:textSize="20sp"
11     android:layout_width="fill_parent"
12     android:layout_height="wrap_content" />
13  <!-- 用于显示文本内容 -->
14  <EditText
15     android:layout_below="@id/TextViewTitle"
16     android:layout_height="wrap_content"
17     android:layout_width="fill_parent"
18     android:id="@+id/EditTextDetail"
19     android:lines="8"
20     android:gravity="top" />
21  <!–“保存”按钮 -->
22  <Button
23     android:layout_height="wrap_content"
24     android:text="@string/button_refer"
25     android:layout_below="@id/EditTextDetail"
26     android:id="@+id/ButtonRefer"
27     android:layout_width="100dp" />
28  <!–“取消”按钮 -->
29  <Button
30     android:layout_height="wrap_content"
31     android:text="@string/button_back"
32     android:layout_below="@id/EditTextDetail"
33     android:layout_toRightOf="@id/ButtonRefer"
34     android:id="@+id/ButtonBack"
35     android:layout_width="100dp" />
36  </RelativeLayout>
```

（8）同样，对于网页文件我们也用自己设计的一个“浏览器”打开，在浏览器中使用了 Android 浏览器的核心组件 WebView，并为这个 WebView 设置了放大和缩小按钮。最下方是一个进度条，将它单独放在一个 RelativeLayout 里面，比较方便控制它的显示和隐藏。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <FrameLayout
03     xmlns:android="http://schemas.android.com/apk/res/android"
04     android:layout_width="fill_parent"
05     android:layout_height="fill_parent"
06     android:background="@drawable/tray_bg">
07     <RelativeLayout
```

```
08          android:id="@+id/weblayout"
09          android:layout width="fill parent"
10          android:layout height="fill parent">
11          <!-- 网页浏览器 -->
12          <android.webkit.WebView
13              android:id="@+id/webkit"
14              android:layout_width="fill_parent"
15              android:layout_height="fill_parent">
16          </android.webkit.WebView>
17          <!-- 放大、缩小按钮 -->
18          <ZoomControls
19              android:id="@+id/zoomControls"
20              android:layout alignParentBottom="true"
21              android:layout alignParentRight="true"
22              android:layout width="wrap content"
23              android:layout height="wrap content"
24              android:layout marginBottom="5dip"
25              android:layout marginRight="5dip"></ZoomControls>
26      </RelativeLayout>
27      <RelativeLayout
28          android:id="@+id/loadingLayout"
29          android:background="#802B2B2B"
30          android:layout_width="fill_parent"
31          android:layout_height="fill_parent">
32          <!-- 进度条 -->
33          <ProgressBar
34              android:id="@+id/progressBar"
35              android:layout centerInParent="true"
36              android:layout width="wrap content"
37              android:layout height="wrap content"></ProgressBar>
38      </RelativeLayout>
```

5.3　功 能 实 现

此时我们假设我们是用户，从一开始单击程序图标开始运行程序，到所有功能都单击一遍，想象我们需要走过的所有流程。一开始，笔者正是以这样的思路来设计程序的，现在，我们再重走一下这个过程。

5.3.1　声明变量

首先，展现在我们眼前的是主界面。我们先定义一些需要用到的变量，比如 List 型变量 mFileName 用于存储当前目录的文件名称，相对应的有 List 型变量 mFilePaths，用于存储这些文件对应的路径。在 onCreate()函数中，首先运行函数 initGridView()初始化菜单视图，紧接着用函数 initMenuListener()为各项菜单绑定监听器，为 ListView 列表绑定长按监听器，然后运行函数 initFileListInfo(mRootPath)，默认显示根目录下的文件。

```
01  public class MainActivity extends ListActivity implements OnItemLong
    ClickListener{
02      //声明成员变量
03      //存放显示的文件列表的名称
04      private List<String> mFileName = null;
```

```
05      //存放显示的文件列表的相对应的路径
06      private List<String> mFilePaths = null;
07      //起始目录"/"
08      private String mRootPath = java.io.File.separator;
09      // SD 卡根目录
10      private String mSDCard = Environment.getExternalStorageDirectory()
        .toString();
11      private String mOldFilePath = "";
12      private String mNewFilePath = "";
13      private String keyWords;
14      //用于显示当前路径
15      private TextView mPath;
16      //用于放置工具栏
17      private GridView mGridViewToolbar;
18      private int[] girdview menu image = {R.drawable.menu phone,R.
        drawable.menu sdcard,
19      R.drawable.menu search,R.drawable.menu create,R.drawable.menu
        palse,R.drawable.menu exit};
20      private String[] gridview_menu_title = {"手机","SD 卡","搜索","创建
        ","粘贴","退出"};
21      //代表手机或 SD 卡，1 代表手机，2 代表 SD 卡
22      private static int menuPosition = 1;
23
24      @Override
25      public void onCreate(Bundle savedInstanceState) {
26          super.onCreate(savedInstanceState);
27          setContentView(R.layout.main);
28          //初始化菜单视图
29          initGridViewMenu();
30          //初始化菜单监听器
31          initMenuListener();
32          //为列表项绑定长按监听器
33          getListView().setOnItemLongClickListener(this);
34          mPath = (TextView)findViewById(R.id.mPath);
35          //程序一开始的时候加载手机目录下的文件列表
36           initFileListInfo(mRootPath);
37      }
```

5.3.2　初始化菜单及绑定监听器

在上面我们提到了初始化菜单视图及绑定监听器，接下去我们来看看具体是如何实现的。如下面的代码所示，在 05 行中设定元素被选中时的背景颜色，07 行设置菜单的背景图片，09 行设置菜单显示的个数。然后为视图设置适配器，如 16 行所示。该适配器的实现在代码 20～35 行，用到了我们一开始定义的两个数组：imageResourceArray 和 menuNameArray。将数组的内容分别存储到一个个的 HashMap 中，并将这些 HashMap 存储到一个 ArrayList 中，再将这两个数组的内容分别映射到布局文件 item_menu.xml 中的图片和文字区域。

在菜单的监听器中分别为每一个图标绑定一个函数，前两个图标分别用来显示各目录的内容和 SD 卡目录的内容。接着分别是搜索、创建文件、粘贴、退出程序。

```
01    /**为 GridView 配置菜单资源*/
02    private void initGridViewMenu(){
```

```
03      mGridViewToolbar = (GridView)findViewById(R.id.file_gridview_
        toolbar);
04      //设置选中时候的背景图片
05        mGridViewToolbar.setSelector(R.drawable.menu_item_selected);
06        //设置背景图片
07        mGridViewToolbar.setBackgroundResource(R.drawable
          .menu_background);
08        //设置列数
09        mGridViewToolbar.setNumColumns(6);
10        //设置居中对齐
11        mGridViewToolbar.setGravity(Gravity.CENTER);
12        //设置水平、垂直间距为 10
13        mGridViewToolbar.setVerticalSpacing(10);
14        mGridViewToolbar.setHorizontalSpacing(10);
15        //设置适配器
16        mGridViewToolbar.setAdapter(getMenuAdapter(gridview_menu_title,
          girdview_menu_image));
17    }
18
19    /**菜单适配器*/
20    private SimpleAdapter getMenuAdapter(String[] menuNameArray,
21          int[] imageResourceArray) {
22      //数组列表用于存放映射表
23      ArrayList<HashMap<String, Object>> mData = new ArrayList<HashMap
        <String, Object>>();
24      for (int i = 0; i < menuNameArray.length; i++) {
25          HashMap<String, Object> mMap = new HashMap<String, Object>();
26          //将 image 映射成图片资源
27          mMap.put("image", imageResourceArray[i]);
28          //将 title 映射成标题
29          mMap.put("title", menuNameArray[i]);
30          mData.add(mMap);
31      }
32      //新建简单适配器，设置适配器的布局文件和映射关系
33      SimpleAdapter mAdapter = new SimpleAdapter(this, mData,R.layout
        .item_menu, new String[] { "image", "title" },new int[] { R.id.item_
        image, R.id.item_text });
34      return mAdapter;
35  }
36
37    /**菜单项的监听*/
38    protected void initMenuListener(){
39      mGridViewToolbar.setOnItemClickListener(new OnItemClick
        Listener(){
40
41          public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
42                  long arg3) {
43              switch(arg2){
44              //回到根目录
45              case 0:
46                  menuPosition = 1;
47                  initFileListInfo(mRootPath);
48                  break;
49              //回到 SD 卡根目录
50              case 1:
51                  menuPosition = 2;
52                  initFileListInfo(mSDCard);
53                  break;
54              //显示搜索对话框
```

```
55              case 2:
56                  searchDilalog();
57                  break;
58              //创建文件夹
59              case 3:
60                  createFolder();
61                  break;
62              //粘贴文件
63              case 4:
64                  palseFile();
65                  break;
66              //退出
67              case 5:
68                  MainActivity.this.finish();
69                  break;
70              }
71          }
72      });
73  }
```

5.3.3　设置长按监听器

下一步是设置长按监听器 onItemLongClick，这里首先还是过滤一下两个特殊的菜单项——返回根目录图标和返回上一级目录图标，然后执行 initItemLongClickListener()函数设置监听器。长按时，正常情况将会弹出一个对话框，如图 5.2 所示，包含复制、重命名、删除这三个操作。但是，这些操作都需要建立在文件可读的基础上，如代码 21 行所示。如果是复制操作，则将复制标志位 isCopy 置为 true，并保存当前的文件名和路径；如果是删除操作，则弹出删除对话框；如果是重命名操作，则弹出重命名对话框。

```
01  //长按列表项的事件监听：对长按需要进行一个控制，当列表中包括“返回根目录”和“返回
    上一级”时，
02  //需要对这两列进行屏蔽
03  public boolean onItemLongClick(AdapterView<?> arg0, View arg1, final int
    position, long arg3) {
04      if(isAddBackUp == true){//说明存在返回根目录和返回上一级两列，接下来要对
                                这两列进行屏蔽
05          if(position != 0 && position != 1){
06              initItemLongClickListener(new File(mFilePaths.get
                (position)));
07          }
08      }
09      if(mCurrentFilePath.equals(mRootPath)||mCurrentFilePath.equals
        (mSDCard)){
10          initItemLongClickListener(new File(mFilePaths.get(position)));
11      }
12      return false;
13  }
14    private String mCopyFileName;
15    private boolean isCopy = false;
16  /**长按文件或文件夹时弹出的带 ListView 效果的功能菜单*/
17  private void initItemLongClickListener(final File file){
18      OnClickListener listener = new DialogInterface.OnClickListener(){
19          //item 的值就是从 0 开始的索引值（从列表的第一项开始）
20          public void onClick(DialogInterface dialog, int item) {
```

```
21              if(file.canRead()){//注意，所有对文件的操作必须是在该文件可读的情
                况下才可以，否则报错
22                  if(item == 0){                  //复制
23                      if(file.isFile()&&"txt".equals((file.getName()
                        .substring(file.getName().lastIndexOf(".")+1, file
                        .getName().length())).toLowerCase())){
24                          Toast.makeText(MainActivity.this, "已复制!",
                            Toast.LENGTH_SHORT).show();
25                          //复制标志位，表明已复制文件
26                          isCopy = true;
27                          //取得复制文件的名字
28                          mCopyFileName = file.getName();
29                          //记录复制文件的路径
30                          mOldFilePath = mCurrentFilePath+java.io.File.
                            separator+mCopyFileName;
31                      }else{
32                          Toast.makeText(MainActivity.this, "对不起,目前只支
                            持复制文本文件!", Toast.LENGTH_SHORT).show();
33                      }
34                  }else if(item == 1){                //重命名
35                      initRenameDialog(file);
36                  }else if(item == 2){                //删除
37                      initDeleteDialog(file);
38                  }
39              }else{
40                  Toast.makeText(MainActivity.this, "对不起，您的访问权限不
                    足!", Toast.LENGTH_SHORT).show();
41              }
42          }
43      };
44      //列表项名称
45      String[] mMenu = {"复制","重命名","删除"};
46      //显示操作选择对话框
47      new AlertDialog.Builder(MainActivity.this)
48                          .setTitle("请选择操作!")
49                          .setItems(mMenu, listener)
50                          .setPositiveButton("取消",null).show();
51  }
```



图 5.2　长按显示对话框

5.3.4　显示指定目录内容

如下代码所示，是显示指定目录内容的函数。当前目录用一个静态变量 mCurrentfilePath 存储，当一个文件路径被传入 initFileListInfo 中时，首先初始化 mFileName 和 mFilePaths 这两个 ArrayList，并判断当前目录是否是手机目录或者 SD 卡根目录。如果不是，则通过 initAddBackUp()函数，在当前界面的上方添加返回根目录和上级目录的按钮。

```
01  //用静态变量存储当前目录路径信息
02    public static String mCurrentFilePath = "";
03    /**根据给定的一个文件夹路径字符串遍历出这个文
04     * 件夹中包含的文件名称并配置到 ListView 列表中*/
05    private void initFileListInfo(String filePath){
06      isAddBackUp = false;
07      mCurrentFilePath = filePath;
08      //显示当前的路径
09      mPath.setText(filePath);
10      mFileName = new ArrayList<String>();
11      mFilePaths = new ArrayList<String>();
12      File mFile = new File(filePath);
13      //遍历出该文件夹路径下的所有文件/文件夹
14      File[] mFiles = mFile.listFiles();
15      //只要当前路径不是手机根目录或者是 sd 卡根目录，则显示"返回根目录"和"返回上一级"
16      if(menuPosition == 1&&!mCurrentFilePath.equals(mRootPath)){
17          initAddBackUp(filePath,mRootPath);
18      }else if(menuPosition == 2&&!mCurrentFilePath.equals(mSDCard)){
19          initAddBackUp(filePath,mSDCard);
20      }
21      /*将所有文件信息添加到集合中*/
22      for(File mCurrentFile:mFiles){
23          mFileName.add(mCurrentFile.getName());
24          mFilePaths.add(mCurrentFile.getPath());
25      }
26      /*适配数据*/
27      setListAdapter(new FileAdapter(MainActivity.this,mFileName,
        mFilePaths));
28    }
29    private boolean isAddBackUp = false;
30    /**根据单击→手机"还是"SD 卡"来加"返回根目录"和"返回上一级"*/
31    private void initAddBackUp(String filePath,String phone sdcard){
32      if(!filePath.equals(phone sdcard)){
33          /*列表项的第一项设置为返回根目录*/
34          mFileName.add("BacktoRoot");
35          mFilePaths.add(phone_sdcard);
36          /*列表项的第二项设置为返回上一级*/
37          mFileName.add("BacktoUp");
38          //回到当前目录的父目录即回到上级
39          mFilePaths.add(new File(filePath).getParent());
40          //将添加返回按键标识位设置为 true
41          isAddBackUp = true;
42      }
```

```
43    }
```

5.3.5　创建文件夹

如下所示是创建文件夹的代码，如代码 13 行、15 行所示，我们用 LayoutInflater 来实现在主视图中插入新布局，并操作新布局里面的元素。在 22～29 行为布局中的按钮绑定监听器，当选项改变时，将标志变量 mChecked 分别改变成 1 或者 2。在代码 35 行设置 AlertDialog 的布局文件为 mLL。当创建文件成功的时候，需要用 initFileListInfo 来更新一下当前的目录内容。运行的效果图如图 5.3 所示。

```
01    private String mNewFolderName = "";
02    private File mCreateFile;
03    private RadioGroup mCreateRadioGroup;
04    private static int mChecked;
05    /**创建文件夹的方法：当用户单击软件下面的创建菜单的时候，是在当前目录下创建一个
      文件夹
06     * 静态变量 mCurrentFilePath 存储的就是当前路径
07     * java.io.File.separator 是 Java 给我们提供的一个 File 类中的静态成员，
08     * 它会根据系统的不同来创建分隔符
09     * mNewFolderName 正是我们要创建的新文件的名称，从 EditText 组件上得到*/
10    private void createFolder(){
11      //用于标识当前选中的是文件或者文件夹
12      mChecked = 2;
13      LayoutInflater mLI = (LayoutInflater)this.getSystemService
       (Context.LAYOUT_INFLATER_SERVICE);
14      //初始化对话框布局
15      final LinearLayout mLL = (LinearLayout)mLI.inflate(R.layout
       .create_dialog, null);
16      mCreateRadioGroup = (RadioGroup)mLL.findViewById(R.id.radiogroup_
       create);
17      final RadioButton mCreateFileButton = (RadioButton)mLL.findViewById
       (R.id.create_file);
18      final RadioButton mCreateFolderButton = (RadioButton)mLL.findView
       ById(R.id.create_folder);
19      //设置默认为创建文件夹
20      mCreateFolderButton.setChecked(true);
21      //为按钮设置监听器
22      mCreateRadioGroup.setOnCheckedChangeListener(new RadioGroup
       .OnCheckedChangeListener(){
23          //当选择改变时触发
24          public void onCheckedChanged(RadioGroup arg0, int arg1) {
25              if(arg1 == mCreateFileButton.getId()){
26                  mChecked = 1;
27              }else if(arg1 == mCreateFolderButton.getId()){
28                  mChecked = 2;
29              }
30          }
31      });
32      //显示对话框
33      Builder mBuilder = new AlertDialog.Builder(MainActivity.this)
34      .setTitle("新建")
35      .setView(mLL)
36      .setPositiveButton("创建", new DialogInterface.OnClickListener(){
37          public void onClick(DialogInterface dialog, int which) {
```

```
38              //获得用户输入的名称
39              mNewFolderName = ((EditText)mLL.findViewById(R.id.new
                filename)).getText().toString();
40              if(mChecked == 1){
41                  try {
42                      mCreateFile = new File(mCurrentFilePath+
43                      java.io.File.separator+mNewFolderName+".txt");
44                      mCreateFile.createNewFile();
45                      //刷新当前目录文件列表
46                      initFileListInfo(mCurrentFilePath);
47                  } catch (IOException e) {
48                      Toast.makeText(MainActivity.this, "文件名拼接出错..!!",
                        Toast.LENGTH_SHORT).show();
49                  }
50              }else if(mChecked == 2){
51                  mCreateFile = new File(mCurrentFilePath+java.io.File
                    .separator+mNewFolderName);
52                  if(!mCreateFile.exists()&&!mCreateFile.isDirectory()
                    &&mNewFolderName.length() != 0){
53                      if(mCreateFile.mkdirs()){
54                          //刷新当前目录文件列表
55                          initFileListInfo(mCurrentFilePath);
56                      }else{
57                        Toast.makeText(MainActivity.this, "创建失败，可能是系
                          统权限不够，root 一下？",
58                          Toast.LENGTH SHORT).show();
59                      }
60                  }else{
61                      Toast.makeText(MainActivity.this, "文件名为空，还是重名
                        了呢？",
62                      Toast.LENGTH SHORT).show();
63                  }
64              }
65          }
66      }).setNeutralButton("取消", null);
67      mBuilder.show();
68  }
```



图 5.3　创建文件夹

5.3.6　重命名文件

如下所示是重命名文件的代码，我们同样使用 LayoutInflater 来展开对话框视图，并操作对话框视图中的界面元素。重命名的时候，需要检查当前目录下是否有同名的文件，如果有重名需要提示用户。否则直接使用 file.renameTo()函数进行重命名，并更新当前目录的内容。重命名界面如图 5.4 所示。

```
01    EditText mET;
02    //显示重命名对话框
03    private void initRenameDialog(final File file){
04      LayoutInflater mLI = LayoutInflater.from(MainActivity.this);
05      //初始化重命名对话框
06      LinearLayout mLL = (LinearLayout)mLI.inflate(R.layout.rename_
        dialog, null);
07      mET = (EditText)mLL.findViewById(R.id.new filename);
08      //显示当前的文件名
09      mET.setText(file.getName());
10      //设置监听器
11      OnClickListener listener = new DialogInterface.OnClickListener(){
12          public void onClick(DialogInterface dialog,int which){
13              String modifyName = mET.getText().toString();
14              final String modifyFilePath = file.getParentFile()
                .getPath()+java.io.File.separator;
15              final String newFilePath = modifyFilePath+modifyName;
16              //判断该新的文件名是否已经在当前目录下存在
17              if(new File(newFilePath).exists()){
18                  if(!modifyName.equals(file.getName())){//把"重命名"操作时
                    没做任何修改的情况过滤掉
19                      //弹出该新命名后的文件已经存在的提示，并提示接下来的操作
20                      new AlertDialog.Builder(MainActivity.this)
21                      .setTitle("提示!")
22                      .setMessage("该文件名已存在，是否要覆盖?")
23                      .setPositiveButton("确定", new DialogInterface.On
                        ClickListener(){
24                          public void onClick(DialogInterface dialog,int
                            which){
25                              file.renameTo(new File(newFilePath));
26                              Toast.makeText(MainActivity.this,
27                          "the file path is "+new File(newFilePath), Toast.
                            LENGTH SHORT).show();
28                              //更新当前目录信息
29                              initFileListInfo(file.getParentFile()
                                .getPath());
30                          }
31                      })
32                      .setNegativeButton("取消", null).show();
33                  }
34              }else{
35                  //文件名不重复时直接修改文件名后再次刷新列表
36                  file.renameTo(new File(newFilePath));
37                  initFileListInfo(file.getParentFile().getPath());
38              }
39          }
40
41      };
```

```
42      //显示对话框
43      AlertDialog renameDialog = new AlertDialog.Builder(MainActivity
       .this).create();
44      renameDialog.setView(mLL);
45      renameDialog.setButton("确定", listener);
46      renameDialog.setButton2("取消", new DialogInterface.OnClick
        Listener(){
47         public void onClick(DialogInterface dialog,int which){
48         //什么都不做，关闭当前对话框
49         }
50      });
51      renameDialog.show();
52    }
```



图 5.4　重命名

5.3.7　删除文件

当选择了删除时，将执行代码如下的函数 initDeleteDialog()，显示对话框供用户确认操作，如果是文件则执行 file.delete()，如果是文件夹则执行 deleteFolder，如图 5.5 所示。删除文件夹时要采用递归删除的方法，因为文件夹中可能还有文件夹，甚至有好几层目录。

```
01   //弹出删除文件/文件夹的对话框
02    private void initDeleteDialog(final File file){
03      new AlertDialog.Builder(MainActivity.this)
04      .setTitle("提示!")
05      .setMessage("您确定要删除该"+(file.isDirectory()?"文件夹":"文件")+"吗?")
06      .setPositiveButton("确定", new DialogInterface.OnClickListener(){
07         public void onClick(DialogInterface dialog,int which){
08            if(file.isFile()){
09               //是文件则直接删除
10               file.delete();
11            }else{
12               //是文件夹则用这个方法删除
```

```
13                  deleteFolder(file);
14              }
15              //重新遍历该文件的父目录
16              initFileListInfo(file.getParent());
17          }
18      })
19      .setNegativeButton("取消", null).show();
20    }
21
22    //删除文件夹的方法（递归删除该文件夹下的所有文件）
23  public void deleteFolder(File folder){
24      File[] fileArray = folder.listFiles();
25      if(fileArray.length == 0){
26          //空文件夹则直接删除
27          folder.delete();
28      }else{
29          //遍历该目录
30          for(File currentFile:fileArray){
31              if(currentFile.exists()&&currentFile.isFile()){
32                  //文件则直接删除
33                  currentFile.delete();
34              }else{
35                  //递归删除
36                  deleteFolder(currentFile);
37              }
38          }
39          folder.delete();
40      }
41  }
```

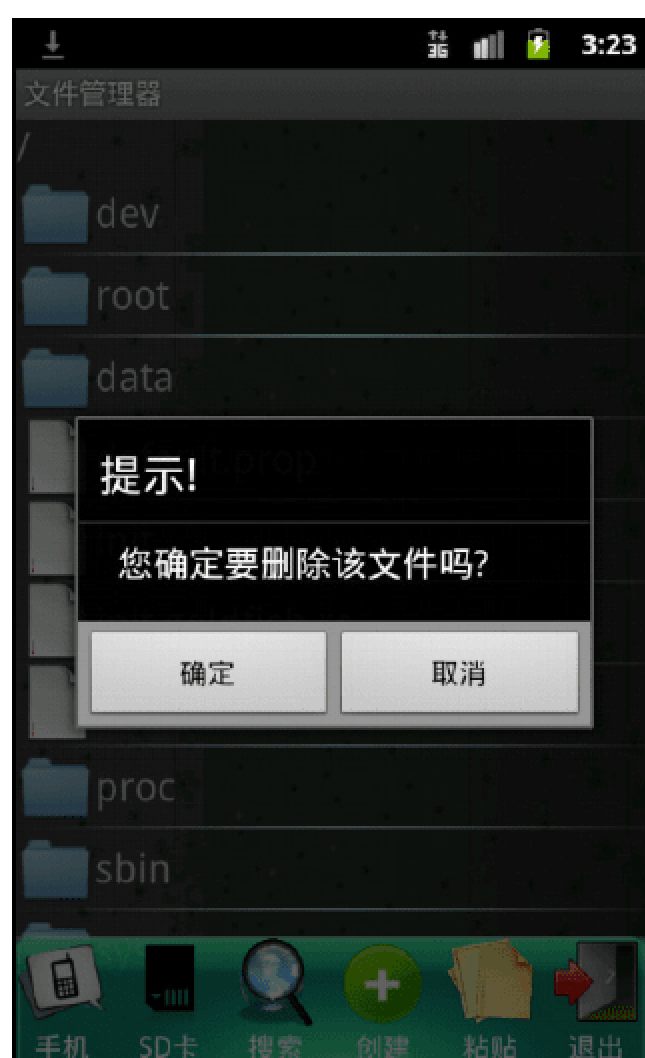


图 5.5　删除文件

5.3.8　粘贴文件

以下代码是粘贴文件的函数，粘贴前需要保证对文件执行过复制操作，这样 inCopy 的值就会变成 true。并且为了保证粘贴是有效的，粘贴的路径要与源文件的路径不一致，

最后要判断目标路径是否也存在同样文件，如代码 06 行所示，如果有相同的文件，则提示用户作出选择“是否要覆盖”。

复制文件的实现函数在代码 26～53 行，通过文件流的方式，将源文件逐个 byte 复制到目标文件中。

```
01 /**粘贴*/
02   private void palseFile(){
03     mNewFilePath = mCurrentFilePath+java.io.File.separator+mCopy
       FileName;                       //得到新路径
04     Log.d("copy", "mOldFilePath is "+mOldFilePath+"| mNewFilePath is
       "+mNewFilePath+"| isCopy is "+isCopy);
05     if(!mOldFilePath.equals(mNewFilePath)&&isCopy == true){
                                       //在不同路径下复制才有效
06         if(!new File(mNewFilePath).exists()){
07             copyFile(mOldFilePath,mNewFilePath);
08             Toast.makeText(MainActivity.this, "执行了粘贴", Toast.LENGTH_
               SHORT).show();
09             initFileListInfo(mCurrentFilePath);
10         }else{
11             new AlertDialog.Builder(MainActivity.this)
12             .setTitle("提示!")
13             .setMessage("该文件名已存在，是否要覆盖?")
14             .setPositiveButton("确定", new DialogInterface.OnClick
               Listener(){
15                 public void onClick(DialogInterface dialog,int which){
16                     copyFile(mOldFilePath,mNewFilePath);
17                     initFileListInfo(mCurrentFilePath);
18                 }
19             })
20             .setNegativeButton("取消", null).show();
21         }
22     }else{
23         Toast.makeText(MainActivity.this, "未复制文件! ", Toast.LENGTH_
           LONG).show();
24     }
25   }
26   private int i;
27   FileInputStream fis;
28   FileOutputStream fos;
29   //复制文件
30   private void copyFile(String oldFile,String newFile){
31     try {
32         fis =  new FileInputStream(oldFile);
33         fos = new FileOutputStream(newFile);
34         do{
35             //逐个 byte 读取文件，并写入另一个文件中
36             if((i = fis.read()) != -1){
37                 fos.write(i);
38             }
39         }while(i != -1);
40         //关闭输入文件流
41         if(fis != null){
42             fis.close();
43         }
44         //关闭输出文件流
45         if(fos != null){
46             fos.close();
```

```
47          }
48      } catch (FileNotFoundException e) {
49          e.printStackTrace();
50      } catch (IOException e) {
51          e.printStackTrace();
52      }
53    }
```

5.3.9　搜索文件

当单击菜单项中的“搜索”按钮时，将调用 serachDialog()函数。和新建文件类似，首先我们要判断当前用户选择的是在当前文件夹搜索还是在整个目录搜索。通过 mRadioChecked 的值可以确定，当 mRadioChecked 的值为 1 时，表明是在当前目录，为 2 时表明是在整个目录下搜索。

当单击“搜索”按钮的时候，将会发送一个广播，将搜索的关键字和将要搜索的目录传递过去，接着开启服务，进行搜索。

```
01  //显示搜索对话框
02    private void searchDilalog(){
03      //用于确定是在当前目录搜索还是在整个目录搜索的标志
04      mRadioChecked = 1;
05      LayoutInflater mLI = LayoutInflater.from(MainActivity.this);
06      final View mLL = (View)mLI.inflate(R.layout.search_dialog, null);
07      mRadioGroup = (RadioGroup)mLL.findViewById(R.id.radiogroup_search);
08      final RadioButton mCurrentPathButton = (RadioButton)mLL.findView
        ById(R.id.radio_currentpath);
09      final RadioButton mWholePathButton = (RadioButton)mLL.findView
        ById(R.id.radio_wholepath);
10      //设置默认选择在当前路径搜索
11      mCurrentPathButton.setChecked(true);
12      mRadioGroup.setOnCheckedChangeListener(new RadioGroup.OnChecked
        ChangeListener(){
13          //当选择改变时触发
14          public void onCheckedChanged(RadioGroup radiogroup,int checkId) {
15              //当前路径的标志为 1
16              if(checkId == mCurrentPathButton.getId()){
17                  mRadioChecked = 1;
18                  //整个目录的标志为 2
19              }else if(checkId == mWholePathButton.getId()){
20                  mRadioChecked = 2;
21              }
22          }
23      });
24      Builder mBuilder = new AlertDialog.Builder(MainActivity.this)
25      .setTitle("搜索").setView(mLL)
26      .setPositiveButton("确定", new OnClickListener(){
27          public void onClick(DialogInterface arg0, int arg1) {
28              keyWords = ((EditText)mLL.findViewById(R.id.edit_search))
                .getText().toString();
29              if(keyWords.length() == 0){
30                  Toast.makeText(MainActivity.this, "关键字不能为空!", Toast
                    .LENGTH_SHORT).show();
31                  searchDilalog();
```

```
32              }else{
33                  if(menuPosition == 1){
34                      mPath.setText(mRootPath);
35                  }else{
36                      mPath.setText(mSDCard);
37                  }
38                  //获取用户输入的关键字并发送广播-开始
39                  Intent keywordIntent = new Intent();
40                  keywordIntent.setAction(KEYWORD_BROADCAST);
41                  //传递搜索的范围区间:1.当前路径下搜索 2.SD 卡下搜索
42                  if(mRadioChecked == 1){
43                      keywordIntent.putExtra("searchpath", mCurrent
                        FilePath);
44                  }else{
45                      keywordIntent.putExtra("searchpath", mSDCard);
46                  }
47                  //传递关键字
48                  keywordIntent.putExtra("keyword", keyWords);
49                  /*到这里为止是携带关键字信息并发送了广播，
50                  会在 Service 服务当中接收该广播并提取关键字进行搜索*/
51                  getApplicationContext().sendBroadcast(keywordIntent);
52                  //获取用户输入的关键字并发送广播-结束
53                  serviceIntent = new Intent("com.android.service.FILE_
                    SEARCH_START");
54                  MainActivity.this.startService(serviceIntent);
                                                  //开启服务，启动搜索
55                  isComeBackFromNotification = false;
56              }
57          }
58      })
59      .setNegativeButton("取消", null);
60      mBuilder.create().show();
61  }
```

5.3.10　接收器源文件

接收器的源文件 SearchBroadCast.java 代码如下所示，定义了两个静态变量：mServiceKeyword 和 mServiceSearchPath，分别用来存储关键字和搜索路径。在 14 行和 15 行，当接收到广播的时候，就对这两个变量赋值。

```
01  package com.supermario.filemanager;            //声明包语句
//02~04 行为引入相关类，这里不再列举，请阅读光盘内容
……
05  public class SearchBroadCast extends BroadcastReceiver {
06      public static  String mServiceKetword = "";//接收搜索关键字的静态变量
07      public static  String mServiceSearchPath = "";//接收搜索路径的静态变量
08      @Override
09      public void onReceive(Context context, Intent intent) {
10          //取得 intent
11          String mAction = intent.getAction();
12          if(MainActivity.KEYWORD_BROADCAST.equals(mAction)){
13              //取得 intent 传递过来的信息
14              mServiceKetword = intent.getStringExtra("keyword");
15              mServiceSearchPath = intent.getStringExtra("searchpath");
16          }
```

```
17      }
18  }
```

5.3.11　搜索服务

以下是搜索服务 FileService 的代码，当用户单击启动这个服务的时候，先后执行了 onCreate()和 onStart()函数。在 onCreate()函数中，代码 031～035 行新建了一个线程用于执行搜索操作。在 onStart()函数中通过执行 fileSearchNotification()函数来发出通知，表明当前正在搜索文件。

我们来看下这个搜索服务是如何执行搜索操作的，在 initFileArray 中首先判断当前文件或文件夹是否可读，如果不加这个判断将会导致异常，如代码 079 行所示。接着遍历目录下的所有文件，并获取文件名与关键字进行比较，如果包含关键字就存储到 mFileName 和 mFilePaths 中。这里还设置了一个变量 m，初值设置为–1，代码 083～088 行对搜索结果作了个处理，当搜索到第一个结果的时候会在上面添加一个“返回到之前目录”的按钮。

由于考虑到文件夹中可能仍有文件夹，因此需要采用递归的方式，如代码 093～098 所示。在每次递归搜索文件夹的开始，需要判断用户是否取消了搜索，如果取消了搜索，就直接停止当前的搜索。

代码 103～115 行实现通知，主要通过 PendingIntent 存储当前的 context 和 intent，然后当用户单击“通知”按钮时可以显示相应的信息。

```
001 package com.supermario.filemanager;                //声明包语句
//002~014 行为引入相关类，这里不再列举，请阅读光盘内容。
……
015 public class FileService extends Service {
016     private Looper mLooper;
017     private FileHandler mFileHandler;
018     private ArrayList<String> mFileName = null;
019     private ArrayList<String> mFilePaths = null;
020     public static final String FILE_SEARCH_COMPLETED = "com.supermario
        .file.FILE_SEARCH_COMPLETED";
021     public static final String FILE NOTIFICATION = "com.supermario
        .file.FILE NOTIFICATION";
022     @Override
023     public IBinder onBind(Intent arg0) {
024         return null;
025     }
026     //创建服务
027     @Override
028     public void onCreate() {
029         super.onCreate();
030         Log.d("FileService", "file service is onCreate");
031         //新建处理线程
032         HandlerThread mHT = new HandlerThread("FileService",Handler
            Thread.NORM PRIORITY);
033         mHT.start();
034         mLooper = mHT.getLooper();
035         mFileHandler = new FileHandler(mLooper);
036     }
037     //服务开始
038     @Override
039     public void onStart(Intent intent, int startId) {
```

```
040         super.onStart(intent, startId);
041         Log.d("FileService", "file service is onStart");
042         mFileName = new ArrayList<String>();
043         mFilePaths = new ArrayList<String>();
044         mFileHandler.sendEmptyMessage(0);
045         //发出通知表明正在进行搜索
046         fileSearchNotification();
047     }
048     @Override
049     public void onDestroy() {
050         super.onDestroy();
051         //取消通知
052         mNF.cancel(R.string.app_name);
053     }
054     class FileHandler extends Handler{
055         public FileHandler(Looper looper){
056             super(looper);
057         }
058         @Override
059         public void handleMessage(Message msg) {
060             super.handleMessage(msg);
061             Log.d("FileService", "file service is handleMessage");
062             //在指定范围搜索
063             initFileArray(new File(SearchBroadCast.mServiceSearch
                Path));
064             //当用户单击了取消搜索则不发送广播
065             if(!MainActivity.isComeBackFromNotification == true){
066                 Intent intent = new Intent(FILE_SEARCH_COMPLETED);
067                 intent.putStringArrayListExtra("mFileNameList", m
                    FileName);
068                 intent.putStringArrayListExtra("mFilePathsList", m
                    FilePaths);
069                 //搜索完毕之后携带数据并发送广播
070                 sendBroadcast(intent);
071             }
072         }
073     }
074     private int m = -1;
075     /**具体做搜索事件的可回调函数*/
076     private void initFileArray(File file){
077         Log.d("FileService", "currentArray is "+file.getPath());
078         //只能遍历可读的文件夹，否则会报错
079         if(file.canRead()){
080             File[] mFileArray = file.listFiles();
081             for(File currentArray:mFileArray){
082                 if(currentArray.getName().indexOf(SearchBroadCast
                    .mServiceKeyword) != -1){
083                     if (m == -1) {
084                         m++;
085                         // 返回搜索之前的目录
086                         mFileName.add("BacktoSearchBefore");
087                         mFilePaths.add(MainActivity.mCurrentFilePath);
088                     }
089                     mFileName.add(currentArray.getName());
090                     mFilePaths.add(currentArray.getPath());
091                 }
092                 //如果是文件夹则回调该方法
093                 if(currentArray.exists()&&currentArray.isDirectory()){
094                     //如果用户取消了搜索，应该停止搜索的过程
095                     if(MainActivity.isComeBackFromNotification == true){
```

```
096                     return;
097                 }
098                 initFileArray(currentArray);
099             }
100         }
101     }
102 }
103 NotificationManager mNF;
104 /**通知*/
105 private void fileSearchNotification(){
106     Notification mNotification = new Notification(R.drawable.logo,
        "后台搜索中...",System.currentTimeMillis());
107     Intent intent = new Intent(FILE NOTIFICATION);
108     //打开 notice 时的提示内容
109     intent.putExtra("notification", "当通知还存在，说明搜索未完成，可以
        在这里触发一个事件，当点击通知回到 Activity 之后，可以弹出一个框，提示是否
        取消搜索！");
110     PendingIntent mPI = PendingIntent.getBroadcast(this,0,
        intent,0);
111     mNotification.setLatestEventInfo(this, "在"+SearchBroadCast.
        mServiceSearchPath+"下搜索", "搜索关键字为"+SearchBroadCast
        .mServiceKeyword+"【点击可取消搜索】", mPI);
112     if(mNF == null){
113         mNF = (NotificationManager)getSystemService(NOTIFICATION
            SERVICE);
114     }
115     mNF.notify(R.string.app name, mNotification);
116 }
117 }
```

5.3.12　广播接收器

在搜索过程中我们用到了两个广播接收器，一个是刚开始执行关键字搜索的时候调用的 SearchBroadCast，另一个是搜索结束后用到的 FileBroadcast。这两个接收器都采用动态注册的方式，在 onStart()函数中进行接收器的注册，在 Destroy 中取消接收器的注册，如代码 20 行、21 行、32 行、33 行所示。

第一个接收器我们在前面已经介绍过了，下面我们看一下 FileBroadCast 的实现方式。可以看出这个接收器根据绑定在 Intent 中的 action 类型可以执行两种不同的操作。当收到的是 FILE_SEARCH_COMPLETED 的 action 时，表明搜索完毕，弹出对话框提示用户是否要显示结果；当收到的 action 是 FILE_NOTIFICATION 时，表明用户单击了通知，此时弹出对话框，询问用户是否取消当前的搜索。

搜索对话框的实现代码在 61~88 行，同样根据当前 action 值的不同显示两种不同的对话框。当搜索完毕，并单击了“确定”按钮时，如果没有任何匹配的结果，则提示“无相关文件/文件夹”；若有结果则为当前 ListView 绑定新的 Adapter。当弹出是否取消搜索的对话框时，用户单击“确定”按钮将会使 isComeBackFromNotification 置为 true，并停止搜索服务，使文件搜索停止。

```
01  /**注册广播*/
02  private IntentFilter mFilter;
03  private FileBroadcast mFileBroadcast;
```

```
04   private IntentFilter mIntentFilter;
05  private SearchBroadCast mServiceBroadCast;
06  @Override
07   protected void onStart() {
08     super.onStart();
09     mFilter = new IntentFilter();
10     mFilter.addAction(FileService.FILE_SEARCH_COMPLETED);
11     mFilter.addAction(FileService.FILE_NOTIFICATION);
12     mIntentFilter = new IntentFilter();
13     mIntentFilter.addAction(KEYWORD_BROADCAST);
14     if(mFileBroadcast == null){
15         mFileBroadcast = new FileBroadcast();
16     }
17     if(mServiceBroadCast == null){
18         mServiceBroadCast = new SearchBroadCast();
19     }
20     this.registerReceiver(mFileBroadcast, mFilter);
21     this.registerReceiver(mServiceBroadCast, mIntentFilter);
22   }
23
24
25  /**注销广播*/
26   @Override
27  protected void onDestroy() {
28     super.onDestroy();
29     Log.d("NullPointError", "onDestroy");
30     mFileName.clear();
31     mFilePaths.clear();
32     this.unregisterReceiver(mFileBroadcast);
33     this.unregisterReceiver(mServiceBroadCast);
34  }
35
36  private String mAction;
37   public static boolean isComeBackFromNotification = false;
38   /**内部广播类*/
39   class FileBroadcast extends BroadcastReceiver{
40     @Override
41     public void onReceive(Context context, Intent intent) {
42         mAction = intent.getAction();
43         //搜索完毕的广播
44         if(FileService.FILE_SEARCH_COMPLETED.equals(mAction)){
45             mFileName = intent.getStringArrayListExtra("mFile
               NameList");
46             mFilePaths = intent.getStringArrayListExtra("mFile
               PathsList");
47             Toast.makeText(MainActivity.this, "搜索完毕!", Toast.LENGTH_
               SHORT).show();
48             //这里搜索完毕之后应该弹出一个弹出框提示用户要不要显示数据
49             searchCompletedDialog("搜索完毕，是否马上显示结果?");
50             getApplicationContext().stopService(serviceIntent);
                     //当搜索完毕的时候停止服务，然后在服务中取消通知
51         //单击通知栏跳转过来的广播
52         }else if(FileService.FILE_NOTIFICATION.equals(mAction)){
                 //单击通知回到当前 Activity，读取其中信息
53             String mNotification = intent.getStringExtra
                ("notification");
54             Toast.makeText(MainActivity.this, mNotification, Toast
               .LENGTH_LONG).show();
55             searchCompletedDialog("你确定要取消搜索吗?");
```

```
56          }
57      }
58    }
59
60    //搜索完毕后单击通知过来时的提示框
61    private void searchCompletedDialog(String message){
62      Builder searchDialog = new AlertDialog.Builder(MainActivity.this)
63      .setTitle("提示")
64      .setMessage(message)
65      .setPositiveButton("确定", new OnClickListener(){
66          public void onClick(DialogInterface dialog,int which) {
67              //当弹出框时，需要对这个"确定"按钮进行一个判断，因为要对不同的情况作不同
                的处理（2 种情况）
68              // 1.搜索完毕
69              // 2.取消搜索
70              if(FileService.FILE_SEARCH_COMPLETED.equals(mAction)){
71                  if(mFileName.size() == 0){
72                      Toast.makeText(MainActivity.this, "无相关文件/文件夹!",
                        Toast.LENGTH_SHORT).show();
73                      setListAdapter(new FileAdapter(MainActivity.this,
                        mFileName,mFilePaths));   //清空列表
74                  }else{
75                      //显示文件列表
76                      setListAdapter(new FileAdapter(MainActivity.this,
                        mFileName,mFilePaths));
77                  }
78              }else{
79                  //设置搜索标志为 true
80                  isComeBackFromNotification = true;
81                  //关闭服务，取消搜索
82                  getApplicationContext().stopService(serviceIntent);
83              }
84          }
85      })
86      .setNegativeButton("取消", null);
87      searchDialog.create();
88      searchDialog.show();
89    }
```

5.3.13　FileAdapter 代码

前面提到搜索完毕之后需要有一个新的 Adapter，这个 Adapter 的实现代码如下所示。可以看到 FileAdapter 继承于 BaseAdapter，在它的构造函数中，初始化各种类型的文件对应的 icon，然后就是重写 BaseAdapter 的函数 getItem、getView 等。

这个类的核心就是实现 getView()函数，用于将获得的文件名数组和相应的路径数组转换成文件类型对应的 icon 加文件名的形式，而判断文件类型的依据就是通过文件的扩展名。

到这里，整个搜索的过程就讲解完毕，搜索的实际效果如图 5.6 和图 5.7 所示。

```
001    //自定义 Adapter 内部类
002    class FileAdapter extends BaseAdapter{
003     //返回键，各种格式的文件的图标
004     private Bitmap mBackRoot;
005     private Bitmap mBackUp;
006     private Bitmap mImage;
```

```
007     private Bitmap mAudio;
008     private Bitmap mRar;
009     private Bitmap mVideo;
010     private Bitmap mFolder;
011     private Bitmap mApk;
012     private Bitmap mOthers;
013     private Bitmap mTxt;
014     private Bitmap mWeb;
015
016     private Context mContext;
017     //文件名列表
018     private List<String> mFileNameList;
019     //文件对应的路径列表
020     private List<String> mFilePathList;
021
022     public FileAdapter(Context context,List<String> fileName,List
        <String> filePath){
023         mContext = context;
024         mFileNameList = fileName;
025         mFilePathList = filePath;
026         //初始化图片资源
027         //返回到根目录
028         mBackRoot = BitmapFactory.decodeResource(mContext.get
            Resources(),R.drawable.back to root);
029         //返回到上一级目录
030         mBackUp = BitmapFactory.decodeResource(mContext.get
            Resources(),R.drawable.back to up);
031         //图片文件对应的 icon
032         mImage = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.image);
033         //音频文件对应的 icon
034         mAudio = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.audio);
035         //视频文件对应的 icon
036         mVideo = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.video);
037         //可执行文件对应的 icon
038         mApk = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.apk);
039         //文本文件对应的 icon
040         mTxt = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.txt);
041         //其他类型文件对应的 icon
042         mOthers = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.others);
043         //文件夹对应的 icon
044         mFolder = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.folder);
045         //zip 文件对应的 icon
046         mRar = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.zip_icon);
047         //网页文件对应的 icon
048         mWeb = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.web browser);
049     }
050     //获得文件的总数
051     public int getCount() {
052         return mFilePathList.size();
053     }
```

```
054     //获得当前位置对应的文件名
055     public Object getItem(int position) {
056         return mFileNameList.get(position);
057     }
058     //获得当前的位置
059     public long getItemId(int position) {
060         return position;
061     }
062     //获得视图
063     public View getView(int position, View convertView, ViewGroup
        viewgroup) {
064         ViewHolder viewHolder = null;
065         if (convertView == null) {
066             viewHolder = new ViewHolder();
067             LayoutInflater mLI = (LayoutInflater)mContext.getSystem
                Service(Context.LAYOUT_INFLATER_SERVICE);
068             //初始化列表元素界面
069             convertView = mLI.inflate(R.layout.list child, null);
070             //获取列表布局界面元素
071             viewHolder.mIV = (ImageView)convertView.findViewById(R.id
                .image_list_childs);
072             viewHolder.mTV = (TextView)convertView.findViewById(R.id
                .text_list_childs);
073             //将每一行的元素集合设置成标签
074             convertView.setTag(viewHolder);
075         } else {
076             //获取视图标签
077             viewHolder = (ViewHolder) convertView.getTag();
078         }
079         File mFile = new File(mFilePathList.get(position).toString());
080         //如果当前单击的是返回根目录
081         if(mFileNameList.get(position).toString().equals
            ("BacktoRoot")){
082             //添加返回根目录的按钮
083             viewHolder.mIV.setImageBitmap(mBackRoot);
084             viewHolder.mTV.setText("返回根目录");
085         }else if(mFileNameList.get(position).toString().equals
            ("BacktoUp")){
086             //添加返回上一级菜单的按钮
087             viewHolder.mIV.setImageBitmap(mBackUp);
088             viewHolder.mTV.setText("返回上一级");
089         }else if(mFileNameList.get(position).toString().equals
             ("BacktoSearchBefore")){
090             //添加返回搜索之前目录的按钮
091             viewHolder.mIV.setImageBitmap(mBackRoot);
092             viewHolder.mTV.setText("返回搜索之前目录");
093         }else{
094             String fileName = mFile.getName();
095             viewHolder.mTV.setText(fileName);
096             if(mFile.isDirectory()){
097                 viewHolder.mIV.setImageBitmap(mFolder);
098             }else{
099                 String fileEnds = fileName.substring(fileName.last
                    IndexOf(".")+1,fileName.length()).toLowerCase();
                                              //取出文件后缀名并转成小写
100                 if(fileEnds.equals("m4a")||fileEnds.equals("mp3")||
                    fileEnds.equals("mid")||fileEnds.equals("xmf")||
                    fileEnds.equals("ogg")||fileEnds.equals("wav")){
101                     viewHolder.mIV.setImageBitmap(mVideo);
```

```
102                 }else if(fileEnds.equals("3gp")||fileEnds.equals("mp4")){
103                     viewHolder.mIV.setImageBitmap(mAudio);
104 }else if(fileEnds.equals("jpg")||fileEnds.equals("gif")||file
    Ends.equals("png")||fileEnds.equals("jpeg")||fileEnds.equals("bmp")){
105                     viewHolder.mIV.setImageBitmap(mImage);
106                 }else if(fileEnds.equals("apk")){
107                     viewHolder.mIV.setImageBitmap(mApk);
108                 }else if(fileEnds.equals("txt")){
109                     viewHolder.mIV.setImageBitmap(mTxt);
110                 }else if(fileEnds.equals("zip")||fileEnds.equals("rar")){
111                     viewHolder.mIV.setImageBitmap(mRar);
112                 }else if(fileEnds.equals("html")||fileEnds.equals("htm")
                 ||fileEnds.equals("mht")){
113                     viewHolder.mIV.setImageBitmap(mWeb);
114                 }else {
115                     viewHolder.mIV.setImageBitmap(mOthers);
116                 }
117             }
118         }
119         return convertView;
120     }
121     //用于存储列表每一行元素的图片和文本
122     class ViewHolder {
123         ImageView mIV;
124         TextView mTV;
125     }
126    }
```



图 5.6　正在搜索

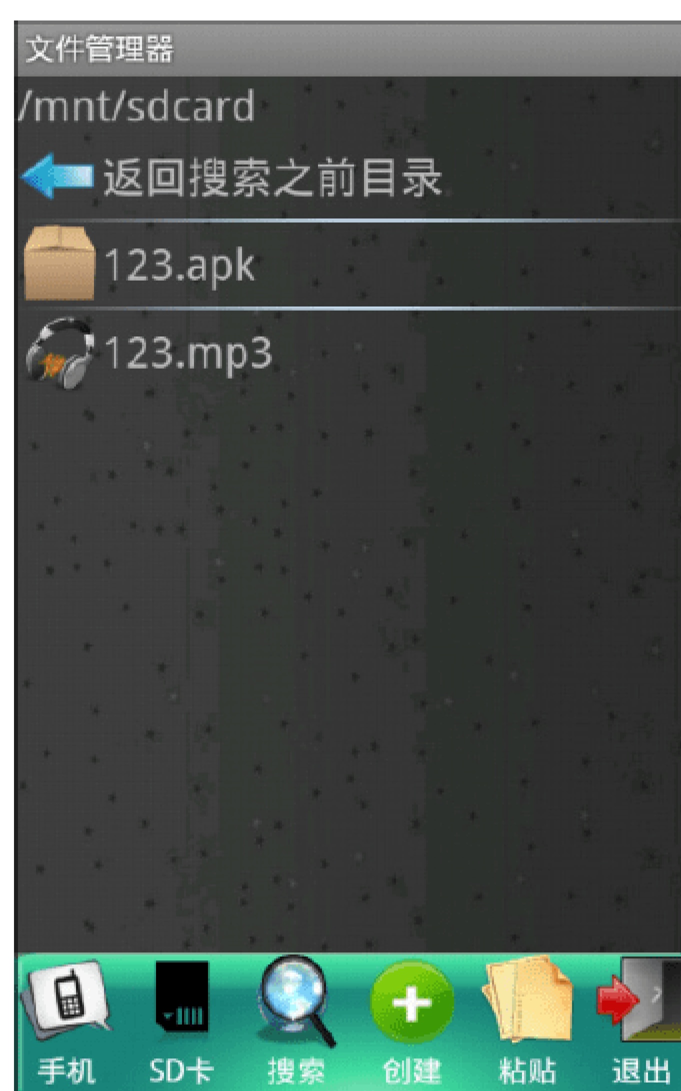


图 5.7　显示搜索结果

5.3.14　打开文件

还有一个功能，也是文件管理器很重要的功能，那就是打开文件。为此，我们需要为 ListView 设置一个按键回调函数，如下代码所示。当用户单击列表中的文件的时候，首先判断是文件还是文件夹，若是文件夹，直接进入文件夹并显示文件目录。若是文件，则根

据文件的类型决定打开方式。其中 txt 文件和 html 文件均使用我们自己的查看工具打开，其他的文件则使用系统的默认打开方式打开。

当打开文本文件的时候，由于需要进行一些比较耗时的工作，所以此时需要显示一个圆形进度条以通知用户等待。这个进度条的实现函数如代码 56～72 行所示，简单设置了标题和可被取消属性。

```
01    /**列表项单击时的事件监听*/
02    @Override
03    protected void onListItemClick(ListView listView, View view, int
      position, long id){
04      final File mFile = new File(mFilePaths.get(position));
05      //如果该文件是可读的，我们进去查看文件
06      if(mFile.canRead()){
07          if(mFile.isDirectory()){
08              //如果是文件夹，则直接进入该文件夹，查看文件目录
09              initFileListInfo(mFilePaths.get(position));
10          }else{
11              //如果是文件，则用相应的打开方式打开
12              String fileName = mFile.getName();
13              String  fileEnds = fileName.substring(fileName.lastIndexOf
                (".")+1,fileName.length()).toLowerCase();
14              if(fileEnds.equals("txt")){
15                  //显示进度条，表示正在读取
16                  initProgressDialog(ProgressDialog.STYLE_HORIZONTAL);
17                  new Thread(new Runnable(){
18                      public void run(){
19                          //打开文本文件
20                          openTxtFile(mFile.getPath());
21                      }
22                  }).start();
23                  new Thread(new Runnable(){
24                      public void run(){
25                          while(true){
26                              if(isTxtDataOk == true){
27                                  //关闭进度条
28                                  mProgressDialog.dismiss();
29                                  executeIntent(txtData.toString(),mFile
                                    .getPath());
30                                  break;
31                              }
32                              if(isCancleProgressDialog == true){
33                                  //关闭进度条
34                                  mProgressDialog.dismiss();
35                                  break;
36                              }
37                          }
38                      }
39                  }).start();
40                  //如果是 html 文件则用自己写的工具打开
41              } else if(fileEnds.equals("html")||fileEnds.equals("mht")
                ||fileEnds.equals("htm")){
42                  Intent intent = new Intent(MainActivity.this,Web
                    Activity.class);
43                  intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
44                  intent.putExtra("filePath", mFile.getPath());
45                  startActivity(intent);
46              } else {
```

```
47                openFile(mFile);
48             }
49          }
50      }else{
51          //如果该文件不可读，我们给出提示不能访问，防止用户操作系统文件造成系统崩溃等
52          Toast.makeText(MainActivity.this, "对不起，您的访问权限不足!",
            Toast.LENGTH_SHORT).show();
53      }
54    }
55  //进度条
56    ProgressDialog mProgressDialog;
57    boolean isCancleProgressDialog = false;
58    /**弹出正在解析文本数据的 ProgressDialog*/
59    private void initProgressDialog(int style){
60      isCancleProgressDialog = false;
61      mProgressDialog = new ProgressDialog(this);
62      mProgressDialog.setTitle("提示");
63      mProgressDialog.setMessage("正在为你解析文本数据，请稍后...");
64      mProgressDialog.setCancelable(true);
65      mProgressDialog.setButton("取消", new DialogInterface.OnClick
        Listener(){
66          public void onClick(DialogInterface arg0, int arg1) {
67              isCancleProgressDialog = true;
68              mProgressDialog.dismiss();
69          }
70      });
71      mProgressDialog.show();
72    }
```

5.3.15　系统默认打开文件的方法

先来看看使用系统默认方式打开文件的方法如何实现，如下代码所示。为了保险起见，我们对文件是否是文件夹又作了一次判断，然后新建一个 intent，为 intent 设置 Flags、Action 和 DataAndType 属性，最后使用函数 startActivity(intent）调用系统方法打开文件。

在设置 DataAndType 属性的时候，需要获得文件的 MIME 类型，该函数的实现如代码 15～30 行所示，也是根据文件扩展名来确定的。作为示例，我们单击一个 jpg 文件，可以看到效果如图 5.8 所示。

```
01    /**调用系统的方法，来打开文件的方法*/
02    private void openFile(File file){
03      if(file.isDirectory()){
04          initFileListInfo(file.getPath());
05      }else{
06          Intent intent = new Intent();
07          intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
08          intent.setAction(android.content.Intent.ACTION_VIEW);
09          //设置当前文件类型
10          intent.setDataAndType(Uri.fromFile(file), getMIMEType(file));
11          startActivity(intent);
12      }
13    }
14    /**获得 MIME 类型的方法*/
15    private String getMIMEType(File file){
16      String type = "";
```

```
17      String fileName = file.getName();
18      //取出文件后缀名并转成小写
19      String  fileEnds = fileName.substring(fileName.lastIndexOf
       (".")+1,fileName.length()).toLowerCase();
20      if(fileEnds.equals("m4a")||fileEnds.equals("mp3")||fileEnds
     .equals("mid")||fileEnds.equals("xmf")||fileEnds.equals("ogg")||
     fileEnds.equals("wav")){
21          type = "audio/*";// 系统将列出所有可能打开音频文件的程序选择器
22      }else if(fileEnds.equals("3gp")||fileEnds.equals("mp4")){
23          type = "video/*";// 系统将列出所有可能打开视频文件的程序选择器
24      }else if(fileEnds.equals("jpg")||fileEnds.equals("gif")||file
     Ends.equals("png")||fileEnds.equals("jpeg")||fileEnds.equals
     ("bmp")){
25          type = "image/*";// 系统将列出所有可能打开图片文件的程序选择器
26      }else{
27          type = "*/*"; // 系统将列出所有可能打开该文件的程序选择器
28      }
29      return type;
30    }
```



图 5.8　打开 jpg 文件

5.3.16　用编辑器打开文本文件

如果打开的文件是文本文件的话，我们希望使用自己设计的一个简单的编辑器打开。首先，新建一个变量 txtData 用于存储文本文件的内容，使用 openTxtFile()函数打开文本文件，并将文本数据存储到 txtData 变量中，如代码 16 行所示。接着执行 excuteIntent()函数使用 intent 打开文本编辑器，并将文本数据、文件路径和标题信息传递过去。

```
01    String txtData = "";
02    boolean isTxtDataOk = false;
03    //打开文本文件的方法，用于读取文件数据
```

```
04    private void openTxtFile(String file){
05      isTxtDataOk = false;
06      try {
07          FileInputStream fis = new FileInputStream(new File(file));
08          StringBuilder mSb = new StringBuilder();
09          int m;
10          //读取文本文件内容
11          while((m = fis.read()) != -1){
12              mSb.append((char)m);
13          }
14          fis.close();
15          //保存读取到的数据
16          txtData = mSb.toString();
17          //读取完毕
18          isTxtDataOk = true;
19      } catch (FileNotFoundException e) {
20          e.printStackTrace();
21      } catch (IOException e) {
22          e.printStackTrace();
23      }
24    }
25    //执行 Intent 跳转的方法
26    private void executeIntent(String data,String file){
27      Intent intent = new Intent(MainActivity.this,EditTxtActivity.class);
28      intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
29      //传递文件的路径、标题和内容
30      intent.putExtra("path", file);
31      intent.putExtra("title", new File(file).getName());
32      intent.putExtra("data", data.toString());
33      //跳转到 EditTxtActivity
34      startActivity(intent);
35    }
```

5.3.17　文本编辑器的实现

文本编辑器的实现代码如下所示，新建一个 EditTxtActivity，主要通过一个 EditText 显示文本内容，并可以在上面做一些编辑操作，最后单击“保存”按钮，实现对文本文件的编辑。

不过这个文本文件编辑器做得有些粗糙，对中文的处理也不好，可以当做是一个文本编辑器的雏形。读者若有兴趣可以对这个编辑器进行改造，设计出自己适用的编辑器。

如图 5.9 所示，是在编辑器打开一个文本文件之后显示的效果图。

```
01  package com.supermario.filemanager;        //声明包语句
02~14 行为引入相关类，这里不再列举，请阅读光盘内容
//……
15  //文本编辑器
16  public class EditTxtActivity extends Activity implements OnClick
     Listener{
17      //显示打开的文本内容
18      private EditText txtEditText;
19      //显示打开的文件名
20      private TextView txtTextTitle;
21      //“保存”按钮
22      private Button txtSaveButton;
```

```
23      //“取消”按钮
24      private Button txtCancleButton;
25      private String txtTitle;
26      private String txtData;
27      private String txtPath;
28      @Override
29      protected void onCreate(Bundle savedInstanceState) {
30          super.onCreate(savedInstanceState);
31          setContentView(R.layout.edit_txt);
32          //初始化界面
33          initContentView();
34          //获得文件路径
35          txtPath = getIntent().getStringExtra("path");
36          //获得文件名
37          txtTitle = getIntent().getStringExtra("title");
38          //获得文本数据
39          txtData = getIntent().getStringExtra("data");
40          try {
41              txtData = new String(txtData.getBytes("ISO-8859-1"),"UTF-8");
            //转码
42          } catch (UnsupportedEncodingException e) {
43              e.printStackTrace();
44          }
45          txtTextTitle.setText(txtTitle);
46          txtEditText.setText(txtData);
47      }
48      /**组件初始化*/
49      private void initContentView(){
50          txtEditText = (EditText)findViewById(R.id.EditTextDetail);
51          txtTextTitle = (TextView)findViewById(R.id.TextViewTitle);
52          txtSaveButton = (Button)findViewById(R.id.ButtonRefer);
53          txtCancleButton = (Button)findViewById(R.id.ButtonBack);
54          //设置“保存”按钮监听器
55          txtSaveButton.setOnClickListener(this);
56          //设置“取消”按钮监听器
57          txtCancleButton.setOnClickListener(this);
58      }
59      /**单击事件监听*/
60      public void onClick(View view) {
61          if(view.getId() == txtSaveButton.getId()){
62              //保存
63              saveTxt();
64          }else if(view.getId() == txtCancleButton.getId()){
65              EditTxtActivity.this.finish();
66          }
67      }
68      /**保存编辑后的文本信息*/
69      private void saveTxt(){
70          try {
71              //取得编辑框内容
72              String newData = txtEditText.getText().toString();
73              BufferedWriter mBW = new BufferedWriter(new FileWriter(new
                File(txtPath)));
74              //写入文件
75              mBW.write(newData,0,newData.length());
76              mBW.newLine();
77              mBW.close();
78              //提示
```

```
79              Toast.makeText(EditTxtActivity.this, "成功保存!", Toast
                .LENGTH_SHORT).show();
80          } catch (IOException e) {
81              Toast.makeText(EditTxtActivity.this, "存储文件时出现了异常!",
                Toast.LENGTH_SHORT).show();
82              e.printStackTrace();
83          }
84          this.finish();
85      }
86  }
```

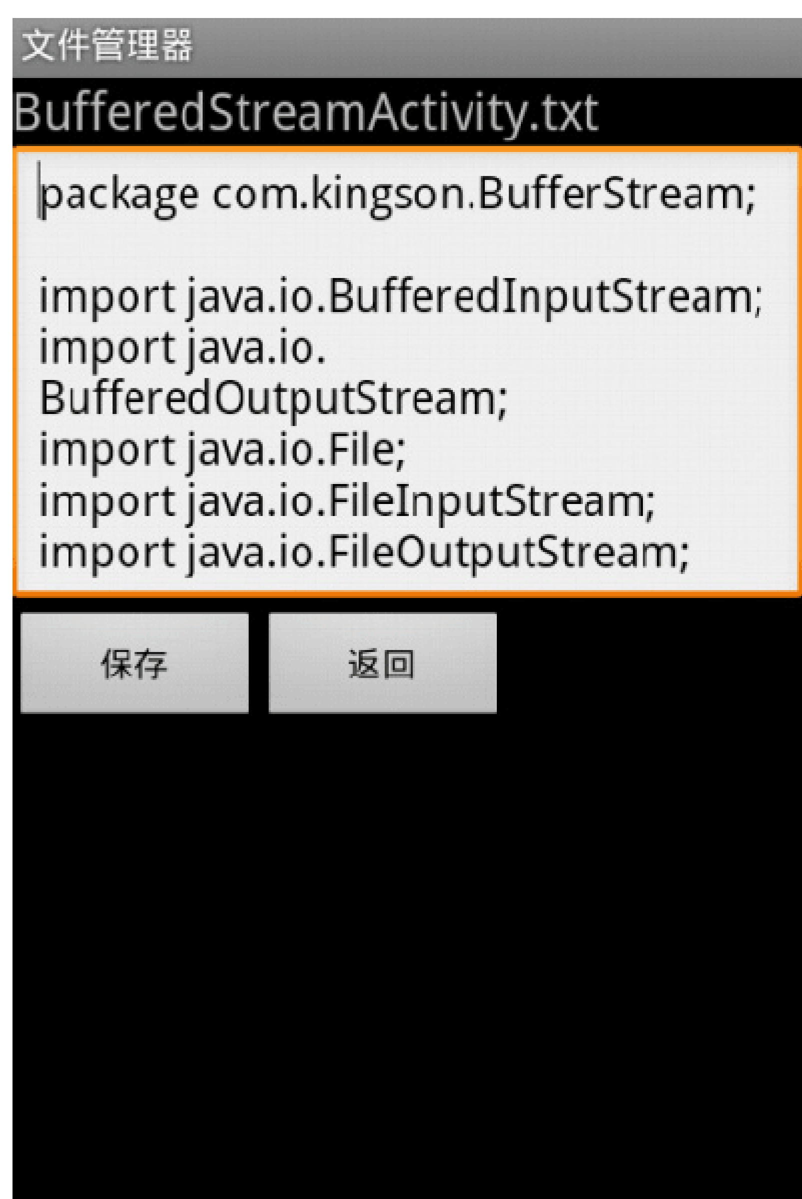


图 5.9　打开文本文档

5.3.18　网页浏览器

除了文本编辑器以外，我们还自己设计了一个简单的网页浏览器。主要是使用到了 WebView，WebView 是个好东西，作用相当于一个迷你的浏览器，采用 Webkit 内核，因此完美支持 HTML、JavaScript、CSS 等。

如下代码所示，在 025～039 行中显示初始化页面的组件，包括 WebView 组件、RelativeLayout 组件等，然后通过 MyAnsyncTask 执行一个异步进程。这个进程一开始先通过 setVisibility 设置布局的可见性，将包含 WebView 的布局隐藏，显示一个进度条，如代码 087 行、088 行所示。

接着这个异步进程在后台执行耗时函数 reading()，如代码 040～057 行所示，webView 通过 loadData() 函数载入网页文件的数据。而网页文件的数据是通过函数 readWebDataToStringFromPath 获得，读取方法也很简单，如代码 064～070 行所示，将文件先转换成 FileInputStream，每次读取 1024 字节，直到读取完成。

当读取文件完成后，首先还是同样的方式，通过函数 setVisibility()隐藏进度条，显示网页主体。同时为浏览器设置放大、缩小功能，如代码 104～118 行所示。最后我们打开测

试用的一个 html 文件，如图 5.10 所示。

```
001 package com.supermario.filemanager;          //声明包语句
002~017 行为引入相关类，这里不再列举，请阅读光盘内容
//……
018 public class WebActivity extends Activity {
019     //网页浏览器
020     private WebView webView;
021     //进度条布局和网页内容主体布局
022     private RelativeLayout loadingLayout,webLayout;
023     //放大、缩小控制器
024     private ZoomControls zoomControls;
025     @Override
026     protected void onCreate(Bundle savedInstanceState) {
027         super.onCreate(savedInstanceState);
028         setContentView(R.layout.web);
029         //初始化页面组件
030         webView = (WebView)findViewById(R.id.webkit);
031         loadingLayout = (RelativeLayout)findViewById(R.id.loading
            Layout);
032         webLayout = (RelativeLayout)findViewById(R.id.weblayout);
033         zoomControls = (ZoomControls)findViewById(R.id.zoomControls);
034         WebSettings webSettings = webView.getSettings();
035         //设置可以使用 js 脚本
036         webSettings.setJavaScriptEnabled(true);
037         //执行异步进程
038         new MyAsyncTask().execute("");
039     }
040     private void reading(){
041         String filePath = getIntent().getStringExtra("filePath");
042         if (filePath != null) {
043             //读取文件
044             webView.loadData(readWebDataToStringFromPath(filePath, new
                FileReadOverBack() {
045                 @Override
046                 public void fileReadOver() {
047                 }
048             }), "text/html", HTTP.UTF 8);
049         } else {
050             new AlertDialog.Builder(WebActivity.this).setTitle("出错了")
                .setMessage("获取文件路径出错！").setPositiveButton("返回",
                new OnClickListener() {
051                 @Override
052                 public void onClick(DialogInterface dialog, int which) {
053                     WebActivity.this.finish();
054                 }
055             });
056         }
057     }
058     //将网页数据读取到一个字符串变量中
059     private String readWebDataToStringFromPath(String path,final File
        ReadOverBack fileReadOverBack){
060         File file = new File(path);
061         StringBuffer stringBuffer = new StringBuffer();
062         try {
063             //读取文件内容
064             FileInputStream inputStream = new FileInputStream(file);
065             byte[] bytes = new byte[1024];
066             int readCount = 0;
```

```
067             while ((readCount = inputStream.read(bytes)) > 0) {
068                 stringBuffer.append(new String(bytes, 0, readCount));
069             }
070             fileReadOverBack.fileReadOver();
071         } catch (FileNotFoundException e) {
072             return "文件不存在!";
073         } catch (IOException e) {
074             return "文件读取错误!";
075         }
076         return stringBuffer.toString();
077     }
078     interface FileReadOverBack{
079         void fileReadOver();
080     }
081     //异步处理类
082     class MyAsyncTask extends AsyncTask<String, String, String>{
083         //首先执行的函数
084         @Override
085         protected void onPreExecute() {
086             super.onPreExecute();
087             loadingLayout.setVisibility(View.VISIBLE);
088             webLayout.setVisibility(View.GONE);
089         }
090         //后台执行
091         @Override
092         protected String doInBackground(String... params) {
093             reading();
094             return null;
095         }
096         @Override
097         protected void onPostExecute(String result) {
098             super.onPostExecute(result);
099             //设置载入进度条隐藏
100             loadingLayout.setVisibility(View.GONE);
101             //设置浏览器内容可见
102             webLayout.setVisibility(View.VISIBLE);
103             //放大按钮
104             zoomControls.setOnZoomInClickListener(new View.OnClick
                Listener() {
105                 //将网页内容放大
106                 @Override
107                 public void onClick(View v) {
108                     webView.zoomIn();
109                 }
110             });
111             //缩小按钮
112             zoomControls.setOnZoomOutClickListener(new View.OnClick
                Listener() {
113                 //将网页内容缩小
114                 @Override
115                 public void onClick(View v) {
116                     webView.zoomOut();
117                 }
118             });
119         }
```

```
120     }
121 }
```

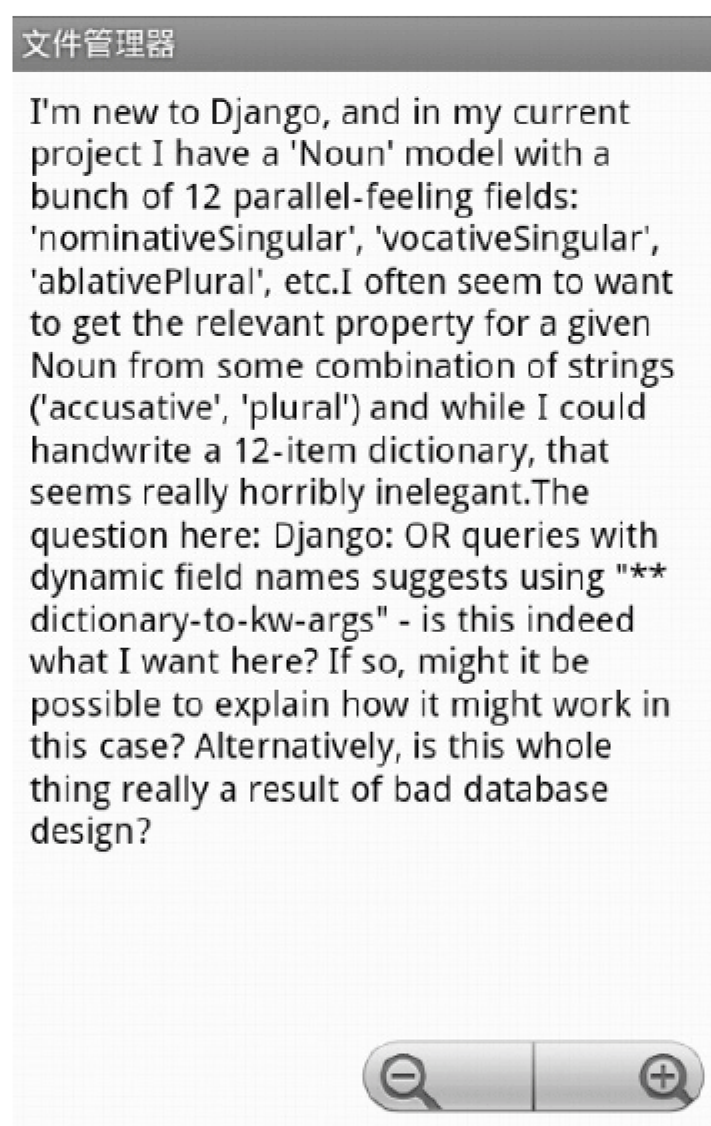


图 5.10　打开 html 文件

5.4　知 识 拓 展

本章在读取 WebView 文件数据的时候采用了一种特别的处理方式——异步操作，Android 提供了一套专门用于异步处理的类——AynsTask 类。使用这个类可以为耗时程序开辟一个新线程进行处理，处理完时返回。其实，AsynTask 类就是对 Thread 类的一个封装，并且加入了一些新的方法。编程时，两者都可以实现同样的功能。本节后面将对 AsynTask 和 Thread 进行比较。

（1）AsynTask 类结构

- doInBackGround()
- onPreExecute()
- onPostExecute()
- onProgressUpdate()

正是这几个回调函数构成了 AsynTask 类的使用逻辑结构，其中每个 AsynTask 子类必须至少复写 doInBackGround()方法。

（2）回调逻辑关系

调用关系如图 5.11 所示。

主线程调用 AsynTask 子类实例的 execute()方法后，首先会调用 onPreExecute()方法。onPreExecute()在主线程中运行，可以用来写一些开始提示代码。之后启动新线程，调用 doInBackground()方法，进行异步数据处理。处理完毕之后异步线程结束，在主线程中调用 onPostExecute()方法。onPostExecute()可以进行一些结束提示处理。

另外，在 doInBackground()方法异步处理的时候，如果希望通知主线程一些数据（如

处理进度），可以调用 publishProgress()方法。这时，主线程会调用 AsynTask 子类的 onProgressUpdate()方法进行处理。

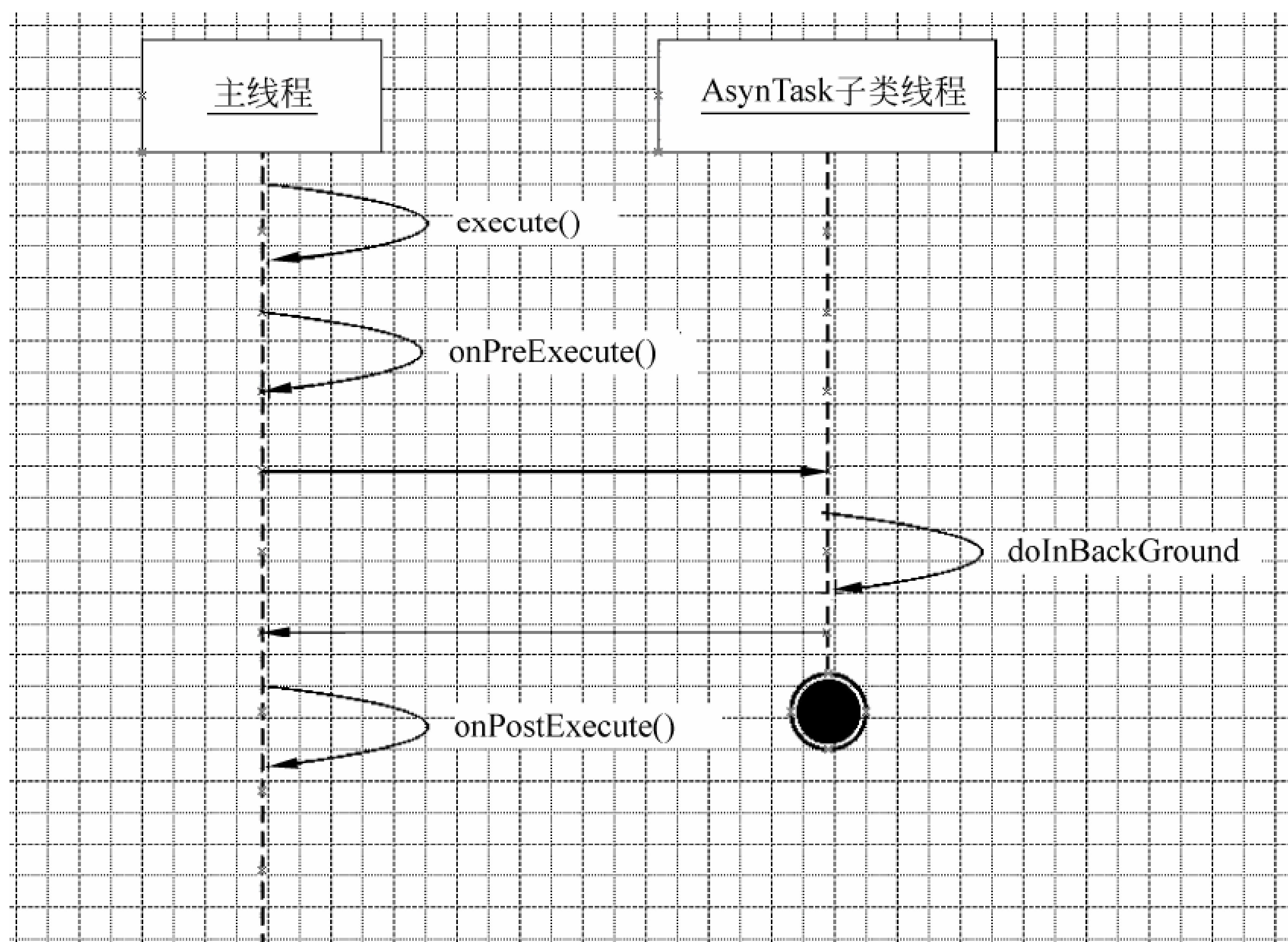


图 5.11　AsynTask 子类线程和主线程回调关系图

（3）各个函数间数据的传递

通过上面的调用关系，我们就可以大概看出一些数据传递关系，如下：execute()向 doInBackground() 传递， doInBackground() 的返回值会传递给 onPostExecute()， publishProgress()向 progressUpdate()传递。

为了使调用关系明确及安全，AsynTask 类在继承时要传入 3 个泛型：第一个泛型对应 execute()向 doInBackground()传递的类型；第二个泛型对应 doInBackground()的返回类型和传递给 onPostExecute()的类型；第三个泛型对应 publishProgress()向 progressUpdate()传递的类型。传递的数据都是对应类型的数组，数组都是可变长的，可以根据具体情况使用。

下面我们看一个简单的例子，例子功能很简单：activity 中有 1 个 textView 和 botton。当单击 botton 时，异步改变 textView 的值，并且在相应的回调函数执行时，用 Log.e 输出值。

在 Eclipse 中新建一个工程 AnsyncTest，界面布局如下所示：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03     android:layout_width="fill_parent"
04     android:layout_height="fill_parent"
05     android:orientation="vertical" >
06     <!-- 用于显示数据 -->
07     <TextView
08        android:id="@+id/text"
09        android:layout_width="fill_parent"
10        android:layout_height="wrap_content"
11        android:text="@string/hello" />
12     <!-- 用于开启异步进程 -->
```

```
13      <Button
14          android:id="@+id/button"
15          android:layout width="fill parent"
16          android:layout height="wrap content"
17          android:text="change"
18          />
```

主体程序如下所示，首先在 onCreate 函数中初始化界面元素，为按键绑定监听器，如代码 18～36 行所示。当用户按下了按键的时候，新建一个异步进程 anys，并执行。当单击按钮的时候，在异步进程类 AnsyTry 中，先后执行 onPreExecute()、doInBackground()、onPostExcute()这 3 个函数，最后更新主界面。我们设置每次单击的时候，在主界面 TextView 的内容后面添加一个“A”，如图 5.12 所示，是我们单击了按钮 4 次之后的结果，对应的 logcat 的信息如图 5.13 所示。

```
01  package com.supermario.ansynctest;                    //声明包语句
02~09 行为引入相关类，这里不再列举，请阅读光盘内容
//……
10  public class AnsyncTestActivity extends Activity {
11      /** Called when the activity is first created. */
12      TextView text =null;
13      Button button=null;
14      String str=null;
15      AnsyTry anys=null;
16      double result=0;
17      @Override
18      public void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.main);
21          //初始化界面元素
22          text=(TextView) findViewById(R.id.text);
23          button=(Button) findViewById(R.id.button);
24          //随意设置一个参数
25          str="flag";
26          //设置监听器
27          button.setOnClickListener(new OnClickListener() {
28              @Override
29              public void onClick(View v) {
30                  // TODO Auto-generated method stub
31                  anys=new AnsyTry(text);
32                  //执行线程
33                  anys.execute(str);
34              }
35          });
36      }
37      class AnsyTry extends AsyncTask<String, TextView, Double>{
38          TextView te=null;
39          public AnsyTry(TextView te) {
40              super();
41              this.te = te;
42          }
43          //执行后台进程
44          @Override
45          protected Double doInBackground(String... params) {
46              // TODO Auto-generated method stub
47              double dou=0;
48              if(params[0].equals("flag")){
49                  //显示当前线程的名称
50                  Log.e("ansync",Thread.currentThread().getName()+"");
```

```
51                  dou=100;
52              }
53              publishProgress(te);
54              return dou;
55          }
56          //后台进程执行完毕
57          @Override
58          protected void onPostExecute(Double result) {
59              // TODO Auto-generated method stub
60              super.onPostExecute(result);
61              Log.e("ansync","postExecute---double---"+result);
62          }
63          //后台进程执行之前
64          @Override
65          protected void onPreExecute() {
66              // TODO Auto-generated method stub\
67              Log.e("ansync","pretExecute------");
68              super.onPreExecute();
69          }
70          //用于更新界面
71          @Override
72          protected void onProgressUpdate(TextView... values) {
73              // TODO Auto-generated method stub
74              //更新 TextView 的内容
75              values[0].setText(values[0].getText()+"A");
76              super.onProgressUpdate(values);
77          }
78      }
79  }
```

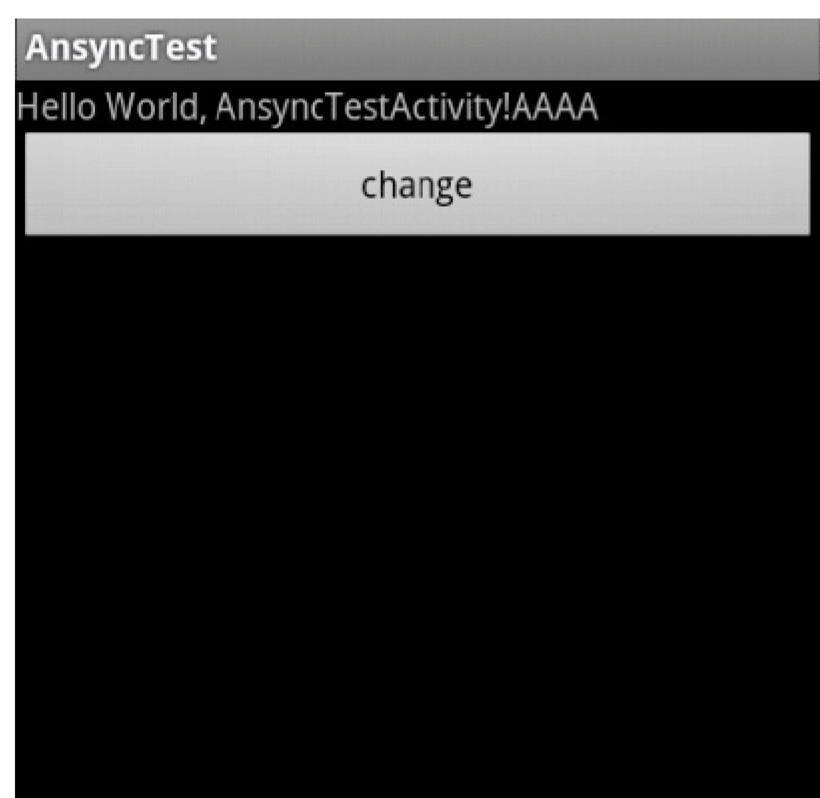


图 5.12　执行异步进程

Application	Tag	Text
com.supermario.ansynctest	ansync	pretExecute------
com.supermario.ansynctest	ansync	AsyncTask #1
com.supermario.ansynctest	ansync	postExecute---double---100.0
com.supermario.ansynctest	ansync	pretExecute------
com.supermario.ansynctest	ansync	AsyncTask #2
com.supermario.ansynctest	ansync	postExecute---double---100.0
com.supermario.ansynctest	ansync	pretExecute------
com.supermario.ansynctest	ansync	AsyncTask #3
com.supermario.ansynctest	ansync	postExecute---double---100.0
com.supermario.ansynctest	ansync	pretExecute------
com.supermario.ansynctest	ansync	AsyncTask #4
com.supermario.ansynctest	ansync	postExecute---double---100.0

图 5.13　执行异步进程的 logcat 信息

5.5　本章小结

本章讲解了如何编写一个简单的文件管理器，要实现一个文件管理器需要具备的一些基本的功能，如创建文件、删除文件、复制文件、打开文件、搜索等，此外针对文本文件和网页文件我们还设计了自己的编辑器和浏览器。本章的布局相对前面几章比较复杂，主要用到了 ListView、GridView 和 WebView 等界面元素，大家必须熟练掌握这些视图组件的使用，才能在今后设计程序的时候游刃有余。

第 6 章　备　忘　录

生活中我们难免会经常忘记一些东西，这时候如果身边有一本备忘录该有多好。其实，我们可以自己开发一个简单的备忘录，让你在百忙之中从容不迫，不会遗漏掉任何重要事情。

6.1　主界面设计

本章要开发的备忘录功能很简单，但是使用起来也很方便，类似于便签纸的功能。一个标题，一个内容加上可以设置闹钟，就基本组成了备忘录所有的功能。备忘录的主界面如图 6.1 所示，最上面一个大大的“添加”按钮用于新建文件，中间显示备忘录的内容。最下面一排按钮用于切换页面。

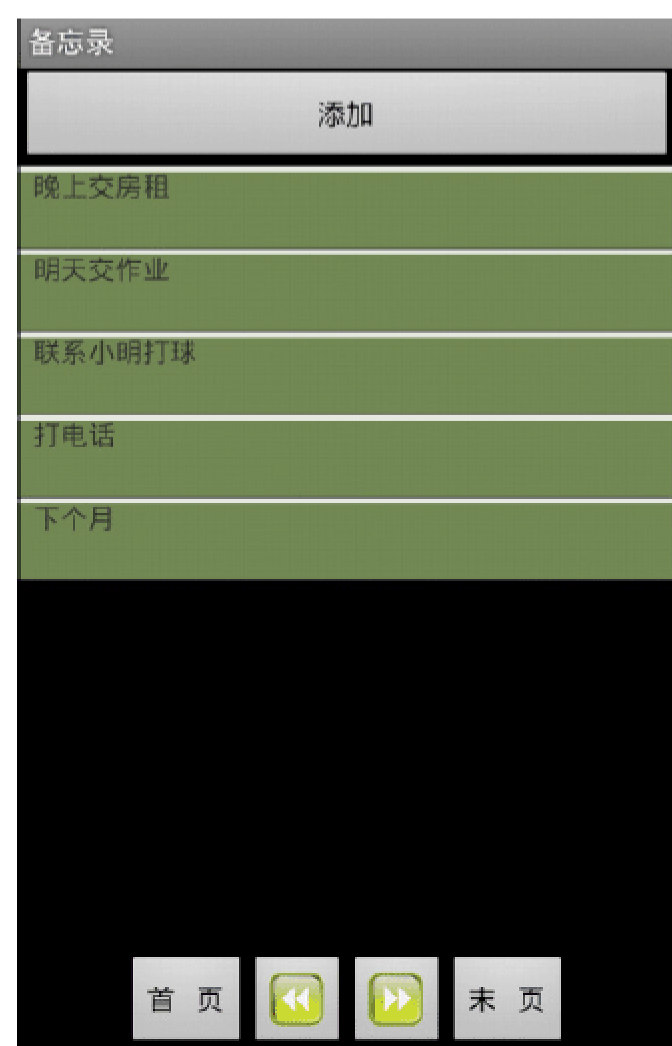


图 6.1　备忘录主界面

主界面设计步骤如下：

（1）在 res/layout 下新建 home.xml，代码如下所示，整个根节点采用 RelativeLayout 布局，最上方采用一个大大的 Button 用于显示“添加”。中间显示数据库中所有的备忘录文件，最下面是一个分页的组件，用一个 Linearlayout 包含 4 个按钮。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03     android:orientation="vertical"
```

```
04      android:layout width="match parent"
05      android:layout height="match parent"
06      >
07      <!-- 添加按钮 -->
08      <Button
09          android:id="@+id/btnAdd"
10          android:layout_width="match_parent"
11          android:layout_height="wrap_content"
12          android:gravity="center_vertical|center_horizontal"
13          android:text="添加"
14      />
15      <!-- 文章列表，用于显示所有备忘录 -->
16      <ListView
17          android:id="@+id/listview"
18          android:layout_width="match_parent"
19          android:layout_height="match_parent"
20          android:layout_below="@+id/btnAdd"
21          android:layout_above="@+id/linearLayout1"
22      />
23      <!-- 底部按钮 -->
24      <LinearLayout
25          android:id="@+id/linearLayout1"
26          android:orientation="horizontal"
27          android:layout width="wrap content"
28          android:layout height="wrap content"
29          android:layout centerHorizontal="true"
30          android:gravity="center"
31          android:layout_alignParentBottom="true"
32          >
33          <!–“首页”按钮 -->
34          <Button
35              android:id="@+id/btnFirst"
36              android:layout width="wrap content"
37              android:layout height="wrap content"
38              android:text="首　页"
39          />
40          <!–“上一页”按钮 -->
41          <ImageButton
42              android:id="@+id/btnPre"
43              android:layout width="wrap content"
44              android:layout height="wrap content"
45              android:src="@drawable/preview"
46          />
47          <!–“下一页”按钮 -->
48          <ImageButton
49              android:id="@+id/btnNext"
50              android:layout_width="wrap_content"
51              android:layout_height="wrap_content"
52              android:src="@drawable/next"
53          />
54          <!–“末页”按钮 -->
55          <Button
56              android:id="@+id/btnEnd"
57              android:layout width="wrap content"
58              android:layout_height="wrap_content"
59              android:text="末　页"
60      />
61          <!-- 进度条 -->
62      </LinearLayout>
```

```
63          <ProgressBar
64          android:id="@+id/progressBar"
65          android:layout_width="wrap_content"
66          android:layout_height="wrap_content"
67          style="?android:attr/progressBarStyleLarge"
68          android:max="100"
69          android:progress="50"
70          android:secondaryProgress="70"
71          android:layout_centerInParent="true"
72          android:visibility="gone"
73          />
74  </RelativeLayout>
```

（2）其中用于匹配 ListView 每一行元素的布局文件如下所示，用一个 TextView 显示备忘录的标题。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:orientation="horizontal"
04      android:layout_width="match_parent"
05      android:layout_height="match_parent"
06      android:background="@drawable/item"
07      >
08      <!-- ListView 元素 -->
09      <TextView
10          android:id="@+id/noteName"
11          android:layout_width="130px"
12          android:layout_height="30px"
13          android:gravity="center_vertical|left"
14          android:textSize="20px"
15          android:layout_marginLeft="10px"
16          android:textColor="#333333"
17      />
18  </LinearLayout>
```

6.2　主界面功能

主界面功能的实现分以下几步：

（1）首先大家需要知道的一点是，我们这个程序是采用 Sqlite 存储数据的，因此我们首先需要新建一个数据库帮助类，用于创建、打开和读写数据库。代码如下所示，新建 SqliteDBConnect.java，并重写 onCrete()和 onUpgrade()函数。

```
01  package com.guo.memorandum;
02
03  import android.content.Context;
04  import android.database.sqlite.SQLiteDatabase;
05  import android.database.sqlite.SQLiteOpenHelper;
06
07  public class SqliteDBConnect extends SQLiteOpenHelper {
08      //创建一个帮助类，用于创建、打开和管理数据库
09      public SqliteDBConnect(Context context) {
10          super(context, "NotePad", null, 1);
11      }
12      //创建数据库，第一次调用的时候执行，之后不再执行
```

```
13      @Override
14      public void onCreate(SQLiteDatabase db) {
15          System.out.println("Table before Create");
16          db.execSQL("create table note(noteId Integer primary key,
            noteName varchar(20),noteTime varchar(20),noteContent varchar
            (400))");
17          System.out.println("Table after Create");
18      }
19      //数据库升级的时候调用
20      @Override
21      public void onUpgrade(SQLiteDatabase db, int oldVersion, int
        newVersion) {
22      }
23  }
```

（2）新建 MainActivity.java，首先声明一些需要用到的变量，如每页显示的数目、数据库帮助类、当前的页码、总页数等。接着在 onCreate 中初始化页面的元素，实例化数据库帮助类。

```
01  public class MainActivity extends Activity {
02      //用于显示备忘录文件
03      private ListView lv;
04      //数据库帮助类
05      private SqliteDBConnect sd;
06      //每页显示的数目
07      private static int page_size = 8;
08      //初始化页数
09      private static int page_no = 1, page_count = 0, count = 0;
10      //“添加”、“首页”、“末页”按钮
11      private Button btnAdd, btnFirst, btnEnd;
12      //图像按钮：前一页、后一页
13      private ImageButton btnNext, btnPre;
14      //适配器
15      private SimpleAdapter sa;
16      //进度条
17      private ProgressBar m_ProgressBar;
18      private ActivityManager am;
19
20      @Override
21      protected void onCreate(Bundle savedInstanceState) {
22          // TODO Auto-generated method stub
23          super.onCreate(savedInstanceState);
24          // 设置显示进度条
25          setProgressBarVisibility(true);
26          setContentView(R.layout.home);
27          //实例化 ActivityManager
28          am = ActivityManager.getInstance();
29          am.addActivity(this);
30          //初始化按钮
31          btnAdd = (Button) findViewById(R.id.btnAdd);
32          btnFirst = (Button) findViewById(R.id.btnFirst);
33          btnPre = (ImageButton) findViewById(R.id.btnPre);
34          btnNext = (ImageButton) findViewById(R.id.btnNext);
35          btnEnd = (Button) findViewById(R.id.btnEnd);
36          //初始化进度条
37          m_ProgressBar = (ProgressBar) findViewById(R.id.progressBar);
38          lv = (ListView) findViewById(R.id.listview);
39          //初始化数据库
```

```
40          sd = new SqliteDBConnect(MainActivity.this);
41          //获取数据库数据并分页显示
```

（3）接着执行分页函数用于获取数据库的数据并显示到页面中，分页函数的实现代码如下所示。如代码 06～17 行所示，先取得数据的总页数。接着根据当前选择的页码，从数据库中取得相应的数据存储到一个个 map 中，然后将所有的 map 存储到 list 中，最后为 ListView 新建一个简单适配器，将 list 中数据适配到 ListView 中。

```
01  //获取数据库数据并分页显示
02  public void fenye() {
03      SQLiteDatabase sdb = sd.getReadableDatabase();
04      count = 0;
05      //从数据库中查询数据，按升序排列
06      Cursor c1 = sdb.query("note", new String[] { "noteId", "noteName",
07              "noteTime" }, null, null, null, null, "noteId asc");
08      while (c1.moveToNext()) {
09          int noteid = c1.getInt(c1.getColumnIndex("noteId"));
10          //保存数据的总数
11          if (noteid > count)
12              count = noteid;
13      }
14      c1.close();
15      //取得总页数
16      page_count = count % page_size == 0 ? count / page_size : count
17              / page_size + 1;
18      //到达首页
19      if (page_no < 1)
20          page_no = 1;
21      //到达末页
22      if (page_no > page_count)
23          page_no = page_count;
24      //查询指定页的数据
25      Cursor c=sdb.rawQuery("select noteId,noteName,noteTime from note
        limit ?,?", new String[] {
26                  (page_no - 1) * page_size + "", page_size + "" });
27      List<Map<String, Object>> list = new ArrayList<Map<String,
        Object>>();
28      //遍历循环，取得所有数据，并存储到 list 中
29      while (c.moveToNext()) {
30          Map<String, Object> map = new HashMap<String, Object>();
31          //取得备忘录的名字
32          String strName = c.getString(c.getColumnIndex("noteName"));
33          //如果字数超过 12 个则去掉后面的字符用...代替
34          if (strName.length() > 20) {
35              map.put("noteName", strName.substring(0, 20) + "...");
36          } else {
37              map.put("noteName", strName);
38          }
39          //取得时间和 id 信息，存储到 map 中
40          map.put("noteTime", c.getString(c.getColumnIndex("noteTime")));
41          map.put("noteId", c.getInt(c.getColumnIndex("noteId")));
42          //将 map 添加到 list 中
43          list.add(map);
44      }
45      c.close();
46      sdb.close();
47      if (count > 0) {
```

```
48          //新建适配器
49          sa = new SimpleAdapter(MainActivity.this, list, R.layout.items,
50                  new String[] { "noteName", "noteTime" }, new int[] {
51                          R.id.noteName, R.id.noteTime });
52          //设置适配器
53          lv.setAdapter(sa);
54      }
55
56  }
```

（4）接着我们设置菜单按钮的功能，重写 onCreateOptionsMenu()函数，并使用 menu.add 添加菜单项“设置铃声”和“退出”。同时需要重写 onMenuItemSelected()函数为菜单项设置相应的功能，当单击“设置铃声”按钮的时候，页面将跳转到设置铃声的页面。关于设置铃声，我们接下去会讲解到，目前先略过。当单击“退出”按钮时，页面就会弹出对话框，询问用户是否确定退出，若确定退出则关闭所有 Activity。此外，当用户想要退出程序的时候也可以按 Back 键，重写 onKeyDown 方法，对 Back 按键进行判断，若是 Back 按键则弹出对话框，让用户选择是否退出本程序。

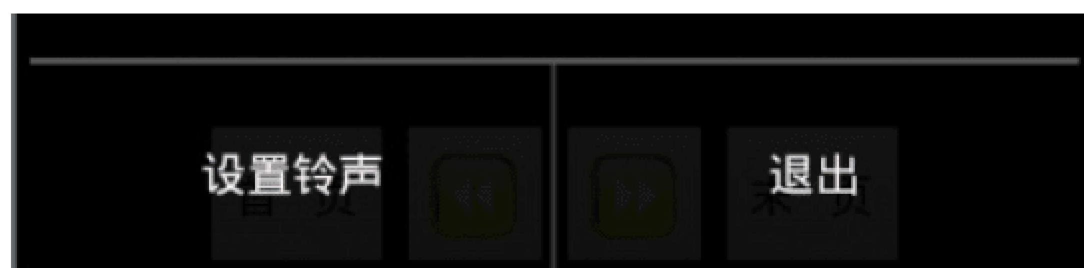


图 6.2　主页面菜单项

```
01  //菜单按钮
02  @Override
03  public boolean onCreateOptionsMenu(Menu menu) {
04      //添加菜单项
05      menu.add(0, 1, 1, "设置铃声");
06      menu.add(0, 2, 2, "退出");
07      return super.onCreateOptionsMenu(menu);
08  }
09  //为菜单按钮绑定按键监听器
10  @Override
11  public boolean onMenuItemSelected(int featureId, MenuItem item) {
12      switch (item.getItemId()) {
13      //设置铃声
14      case 1:
15          Intent intent=new Intent();
16          intent.setClass(MainActivity.this,SetAlarm.class);
17          //跳转到设置铃声的界面
18          startActivity(intent);
19          break;
20      //退出
21      case 2:
22          AlertDialog.Builder adb2 = new Builder(MainActivity.this);
23          adb2.setTitle("消息");
24          adb2.setMessage("真的要退出吗?");
25          adb2.setPositiveButton("确定", new DialogInterface.OnClick
            Listener() {
26              @Override
27              public void onClick(DialogInterface dialog, int which) {
28                  //关闭所有的 Activity
29                  am.exitAllProgress();
```

```
30              }
31          });
32          adb2.setNegativeButton("取消", null);
33          //显示对话框，询问用户是否确定要退出
34          adb2.show();
35          break;
36      default:
37          break;
38      }
39      return super.onMenuItemSelected(featureId, item);
40  }
41  //当用户按键时触发
42  @Override
43  public boolean onKeyDown(int keyCode, KeyEvent event) {
44      //如果用户按下了 Back 键
45      if (keyCode == KeyEvent.KEYCODE_BACK) {
46          AlertDialog.Builder adb = new Builder(MainActivity.this);
47          adb.setTitle("消息");
48          adb.setMessage("真的要退出？");
49          adb.setPositiveButton("确定", new DialogInterface.OnClick
            Listener() {
50              @Override
51              public void onClick(DialogInterface dialog, int which) {
52                  am.exitAllProgress();
53              }
54          });
55          adb.setNegativeButton("取消", null);
56          //显示对话框询问用户是否确定要退出
57          adb.show();
58      }
59      return super.onKeyDown(keyCode, event);
60  }
```

（5）最后，我们要为各个按键绑定监听器。首先是单击列表选项时将进入 Lookover 查看备忘录信息，如代码 002～016 行所示，长按列表选项时将弹出对话框并提供“删除”和“修改”两个选项，如代码 018～063 行所示。065～076 行用于为“添加”按钮设置按键监听器，当单击“添加”按钮时，页面将跳转到 AddActivity。078～139 行用于设置分页相关按钮的功能。

```
001 //设置 ListView 按键监听器
002 lv.setOnItemClickListener(new OnItemClickListener() {
003     @Override
004     public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
005             long arg3) {
006         @SuppressWarnings("unchecked")
007         Map<String, Object> map = (Map<String, Object>) arg0
008                 .getItemAtPosition(arg2);
009         Intent intent = new Intent();
010         //传递备忘录的 noteId
011         intent.putExtra("noteId", map.get("noteId").toString());
012         intent.setClass(MainActivity.this, Lookover.class);
013         //查看备忘录
014         startActivity(intent);
015     }
016 });
017 //设置 ListView 长按监听器
018 lv.setOnItemLongClickListener(new OnItemLongClickListener() {
```

```
019     @Override
020     public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
021             int arg2, long arg3) {
022         @SuppressWarnings("unchecked")
023         final Map<String, Object> map = (Map<String, Object>) arg0
024                 .getItemAtPosition(arg2);
025         AlertDialog.Builder adb = new Builder(MainActivity.this);
026         adb.setTitle(map.get("noteName").toString());
027         //设置弹出选项
028         adb.setItems(new String[] { "删除", "修改"},
029                 new DialogInterface.OnClickListener() {
030                     @Override
031                     public void onClick(DialogInterface dialog,
032                             int which) {
033                         switch (which) {
034                         //删除
035                         case 0:
036                             SQLiteDatabase sdb = sd
037                                     .getReadableDatabase();
038                             sdb.delete("note", "noteId=?",
039                                     new String[] { map.get("noteId")
040                                             .toString() });
041                             Toast.makeText(MainActivity.this, "删除成功",
042                                     Toast.LENGTH_SHORT).show();
043                             sdb.close();
044                             //刷新页面
045                             fenye();
046                             break;
047                         //修改
048                         case 1:
049                             Intent intent = new Intent();
050                             intent.putExtra("noteId", map.get("noteId")
051                                     .toString());
052                             intent.setClass(MainActivity.this,
                                AddActivity.class);
053                             //进入编辑页面
054                             startActivity(intent);
055                             break;
056                         }
057                     }
058                 });
059         //显示对话框
060         adb.show();
061         return true;
062     }
063 });
064 //设置“添加”按钮监听器
065 btnAdd.setOnClickListener(new OnClickListener() {
066     @Override
067     public void onClick(View v) {
068         //显示进度条
069         m_ProgressBar.setVisibility(View.VISIBLE);
070         m_ProgressBar.setProgress(0);
071         Intent intent = new Intent();
072         intent.setClass(MainActivity.this, AddActivity.class);
073         //进入“添加”页面
074         startActivity(intent);
075     }
076 });
```

```
077 //进入首页
078 btnFirst.setOnClickListener(new OnClickListener() {
079     @Override
080     public void onClick(View v) {
081         //如果是首页，提示用户当前已经是首页了
082         if (page_no == 1) {
083             Toast.makeText(MainActivity.this, "已经是首页了", Toast
                .LENGTH_SHORT)
084                     .show();
085         } else {
086             //如果不是首页则将当前页码置为 1
087             page_no = 1;
088         }
089         //刷新页面
090         fenye();
091     }
092 });
093 //设置"下一页"按钮监听器
094 btnNext.setOnClickListener(new OnClickListener() {
095     @Override
096     public void onClick(View v) {
097         //如果当前是最后一页，则提示用户已经到最后一页了
098         if (page_no == page_count) {
099             Toast.makeText(MainActivity.this, "已经是末页了", Toast
                .LENGTH_SHORT)
100                     .show();
101         } else {
102             //否则，当前的页码加 1
103             page_no += 1;
104         }
105         //刷新页面
106         fenye();
107     }
108 });
109 //设置"上一页"按钮监听器
110 btnPre.setOnClickListener(new OnClickListener() {
111     @Override
112     public void onClick(View v) {
113         //如果当前是第一页，则提示用户当前已经是首页了
114         if (page_no == 1) {
115             Toast.makeText(MainActivity.this, "已经是首页了", Toast
                .LENGTH_SHORT)
116                     .show();
117         } else {
118             //否则，当前页码减 1
119             page_no -= 1;
120         }
121         //刷新页面
122         fenye();
123     }
124 });
125 //设置“末页”按钮监听器
126 btnEnd.setOnClickListener(new OnClickListener() {
127     @Override
128     public void onClick(View v) {
129         // TODO Auto-generated method stub
130         //如果当前是最后一页，提示用户当前已经是末页了
131         if (page_no == page_count) {
```

```
132             Toast.makeText(MainActivity.this, "已经是末页了", Toast
                .LENGTH_SHORT)
133                     .show();
134         } else {
135             //否则将当前页置为末页
136             page_no = page_count;
137         }
138         //刷新页面
139         fenye();
140     }
```

6.3　添加和更新备忘录页面

如图 6.3 所示为添加备忘录的页面，从上到下分为四部分：标题、闹钟、内容和功能按钮。

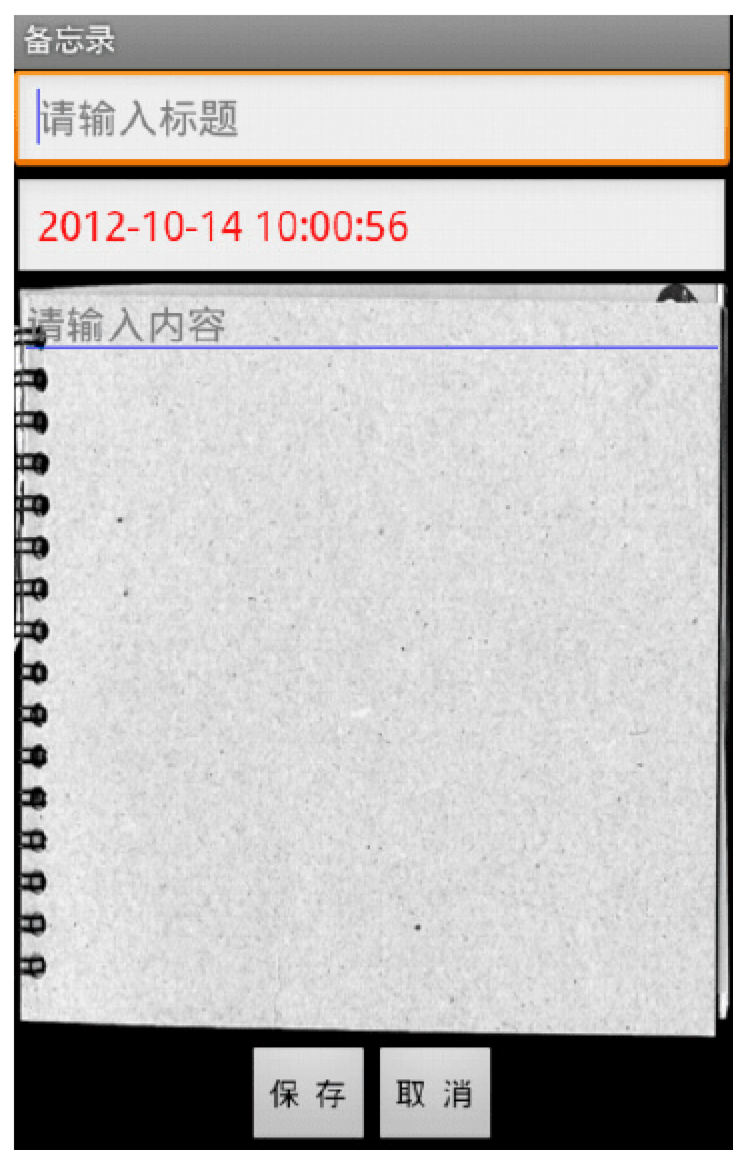


图 6.3　添加备忘录页面

如下所示，是添加页面的代码实现部分，标题和闹钟采用 EditText，其中闹钟为了保证格式的正确，将其设置成不可编辑，而通过其他方式去改变闹钟的时间。view 采用我们自定义的视图类 com.guo.memorandum.LinedEditText 实现，接下去会作进一步分析，最下面两个按钮分别用于实现保存和取消的功能。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout_width="match_parent"
04      android:layout_height="match_parent"
05      android:orientation="vertical" >
06      <!-- 标题 -->
07      <EditText
08          android:id="@+id/noteName"
```

```
09          android:layout width="match parent"
10          android:layout height="wrap content"
11          android:hint="请输入标题"
12          android:textColor="#0000ff" />
13      <!-- 闹钟时间 -->
14      <EditText
15          android:id="@+id/noteTime"
16          android:layout width="match parent"
17          android:layout height="wrap content"
18          android:layout below="@+id/noteName"
19          android:editable="false"
20          android:textColor="#0000ff" />
21      <!-- 备忘录内容 -->
22      <view
23          xmlns:android="http://schemas.android.com/apk/res/android"
24          android:id="@+id/noteMain"
25          android:layout_width="match_parent"
26          android:layout_height="match_parent"
27          android:layout below="@+id/noteTime"
28          android:layout above="@+id/relativeLayout1"
29          class="com.guo.memorandum.LinedEditText"
30          android:background="@drawable/background"
31          android:capitalize="sentences"
32          android:fadingEdge="vertical"
33          android:gravity="top"
34          android:padding="5dip"
35          android:scrollbars="vertical"
36          android:hint="请输入内容"
37          android:textColor="#0000ff" />
38      <!-- 底部按钮 -->
39      <RelativeLayout
40          android:id="@+id/relativeLayout1"
41          android:layout width="wrap content"
42          android:layout height="wrap content"
43          android:layout alignParentBottom="true"
44          android:layout centerHorizontal="true" >
45          <!–“保存”按钮 -->
46          <Button
47              android:id="@+id/btnCommit"
48              android:layout_width="wrap_content"
49              android:layout_height="wrap_content"
50              android:text="保  存" />
51          <!–“取消”按钮 -->
52          <Button
53              android:id="@+id/btnCancel"
54              android:layout_width="wrap_content"
55              android:layout_height="wrap_content"
56              android:layout_toRightOf="@+id/btnCommit"
57              android:text="取  消" />
58      </RelativeLayout>
59  </RelativeLayout>
```

如下所示是上面 view 的实现代码，通过继承于 EditText，自定义一些风格，如文字颜色变成蓝色，为每一行文字增加下划线等。

```
01  package com.guo.memorandum;             //声明包语句
//02~10 行为引入相关类，这里不再列举，请阅读光盘内容
……
11  public class LinedEditText extends EditText {
```

```
12      private Rect mRect;
13      private Paint mPaint;
14      //构造函数
15      public LinedEditText(Context context, AttributeSet attrs) {
16          super(context, attrs);
17          mRect = new Rect();
18          //设置颜料颜色为蓝色
19          mPaint = new Paint();
20          mPaint.setColor(Color.BLUE);
21      }
22      //生成视图
23      @Override
24      protected void onDraw(Canvas canvas) {
25          int count = getLineCount();
26          Rect r = mRect;
27          Paint paint = mPaint;
28          //设置每一行的格式
29          for (int i = 0; i < count; i++) {
30              //取得每一行的基准 Y 坐标，并将每一行的界限值填写到 r 中
31              int baseline = getLineBounds(i, r);
32              //设置每一行的文字带下划线
33              canvas.drawLine(r.left, baseline + 5, r.right, baseline + 5,
                paint);
34          }
35          super.onDraw(canvas);
36      }
```

6.4　添加和更新备忘录功能实现

（1）这部分的核心功能主要分为两部分，一个是闹钟的设定，一个是内容的保存。当然，一如既往地，我们需要先声明一些变量以及初始化界面元素，代码如下所示。代码 15 行声明了一个编辑模式的标志变量 EDIT，默认值为 false，即非编辑模式。当获得的 noteId 变量不为 null 时，说明目前正在进入的是编辑模式，将 EDIT 变量设置为 true，并从数据库中取得相应的信息，填入对应的文本框中。

```
01  public class AddActivity extends Activity {
02      //标题、内容和时间
03      private EditText etName,etMain, etTime;
04      //“保存”按钮、“取消”按钮
05      private Button btnCommit,btnCancel;
06      //数据库操作类
07      private SQLiteDatabase sdb;
08      private ActivityManager am;
09      //年月日时分秒，用于保存日历详细信息
10      private int year, month, day, hours, minute, second;
11      private Calendar c;
12      private PendingIntent pi;
13      private AlarmManager alm;
14      //编辑模式标志
15      private boolean EDIT=false;
16      private String noteId;
17      //初始化函数
18      /** Called when the activity is first created. */
```

```
19      @Override
20      public void onCreate(Bundle savedInstanceState) {
21          super.onCreate(savedInstanceState);
22          setContentView(R.layout.add);
23          //将当前 Activity 添加到 Activity 列表中
24          am = ActivityManager.getInstance();
25          am.addActivity(this);
26          //初始化各个元素
27          etName = (EditText) findViewById(R.id.noteName);
28          etMain = (EditText) findViewById(R.id.noteMain);
29          btnCommit = (Button) findViewById(R.id.btnCommit);
30          btnCancel = (Button) findViewById(R.id.btnCancel);
31          etTime = (EditText) findViewById(R.id.noteTime);
32          Intent intent=getIntent();
33          noteId = intent.getStringExtra("noteId");
34          //如果 noteId 值不为空，则进入编辑模式
35          if(noteId != null)
36              EDIT=true;
37          else
38              EDIT=false;
39          //数据库连接类
40          SqliteDBConnect sd = new SqliteDBConnect(AddActivity.this);
41          //获得数据库操作类
42          sdb = sd.getReadableDatabase();
43          if(EDIT)
44          {
45              //通过 noteId 取得对应的信息
46              Cursor c = sdb.query("note", new String[] { "noteId",
                "noteName",
47                      "noteContent", "noteTime" }, "noteId=?",
48                      new String[] { noteId }, null, null, null);
49              //将获得的信息写入对应的 EditText
50              while (c.moveToNext()) {
51                  etName.setText(c.getString(c.getColumnIndex("noteName")));
52                  etMain.setText(c.getString(c.getColumnIndex("note
                    Content")));
53                  etTime.setText(c.getString(c.getColumnIndex("noteTime")));
54              }
55              c.close();
56          }else{
57              //设置默认闹钟为当前时间
58              etTime.setText(am.returnTime());
59          }
60
61          //设置文本颜色为红色
```

（2）接下去要为界面中一些可单击的地方设置监听器，首先是日期和时间的设定。如代码 001～110 行所示，分别为时间显示区域设置长按监听器和单击监听器，这两种方式类似，下面就以日期为例讲解一下吧。

首先获得当前的日历，并分别提取时分秒的信息，接着新建一个 DatePickerDialog 用于选择日期，默认日期显示为当前日期。当用户设置好日期后，将触发 onDateSet()函数，在该函数中将当前日期更新到对应的 EditText 中。

单击“保存”按钮和“取消”按钮均会弹出对话框让用户进一步选择，若确定保存则调用 saveNote()函数保存备忘录，若取消保存则直接进入主页面并关闭当前页面。

```
001 //为闹钟设置长按监听器，弹出日期选择界面
```

```
002 etTime.setOnLongClickListener(new OnLongClickListener() {
003     @Override
004     public boolean onLongClick(View v) {
005         //实例化日历
006         c = Calendar.getInstance();
007         //取得日历信息中的年月日时分秒
008         year = c.get(Calendar.YEAR);
009         month = c.get(Calendar.MONTH);
010         day = c.get(Calendar.DAY_OF_MONTH);
011         hours = c.get(Calendar.HOUR);
012         minute = c.get(Calendar.MINUTE);
013         second = c.get(Calendar.SECOND);
014         //新建一个日期选择控件
015         DatePickerDialog dpd = new DatePickerDialog(AddActivity.this,
016                 new DatePickerDialog.OnDateSetListener() {
017                     //设置日期的时候触发
018                     @Override
019                     public void onDateSet(DatePicker view, int y,
020                             int monthOfYear, int dayOfMonth) {
021                         String[] time = { "",
022                                 hours + ":" + minute + ":" + second };
023                         try {
024                             //将日期和时间分割
025                             String[] time2 = etTime.getText()
026                                     .toString().trim().split(" ");
027                             //取得时间的信息保存到 time[1]中
028                             if (time2.length == 2) {
029                                 time[1] = time2[1];
030                             }
031                         } catch (Exception e) {
032                             // TODO Auto-generated catch block
033                             e.printStackTrace();
034                         }
035                         String mo = "", da = "";
036                         //将月份转换成两位数
037                         if (monthOfYear < 9) {
038                             mo = "0" + (monthOfYear + 1);
039                         } else {
040                             mo = monthOfYear+1 + "";
041                         }
042                         //将天数转换成两位数
043                         if (dayOfMonth < 10) {
044                             da = "0" + dayOfMonth;
045                         } else {
046                             da = dayOfMonth + "";
047                         }
048                         //将设置的结果保存到 etTime 中
049                         etTime.setText(y + "-" + mo + "-" + da + " "
050                                 + time[1]);
051                     }
052                 }, year, month, day);
053         dpd.setTitle("设置日期");
054         //显示日期控件
055         dpd.show();
056         return true;
057     }
058 });
059 //设置单击监听器，弹出时间选择界面
060 etTime.setOnClickListener(new OnClickListener() {
```

```
061     @Override
062     public void onClick(View v) {
063         //实例化日历
064         c = Calendar.getInstance();
065         //取得当前的年月日信息
066         year = c.get(Calendar.YEAR);
067         month = c.get(Calendar.MONTH);
068         day = c.get(Calendar.DAY_OF_MONTH);
069         //注意这里不是 HOUR，HOUR 返回的是 12 制的时间格式
070         hours = c.get(Calendar.HOUR_OF_DAY);
071         minute = c.get(Calendar.MINUTE);
072         second = c.get(Calendar.SECOND);
073         //新建时间选择器
074         TimePickerDialog tpd = new TimePickerDialog(AddActivity.this,
075                 new OnTimeSetListener() {
076                     @Override
077                     public void onTimeSet(TimePicker view,
078                             int hourOfDay, int minute) {
079                         String[] time = {
080                                 year + "-" + month + "-" + day, "" };
081                         try {
082                             //分割时间和日期
083                             time = etTime.getText().toString().trim()
084                                     .split(" ");
085                         } catch (Exception e) {
086                             // TODO Auto-generated catch block
087                             e.printStackTrace();
088                         }
089                         String ho = "", mi = "";
090                         //设置小时
091                         if (hourOfDay < 10) {
092                             ho = "0" + hourOfDay;
093                         } else {
094                             ho = hourOfDay + "";
095                         }
096                         //设置分钟
097                         if (minute < 10) {
098                             mi = "0" + minute;
099                         } else {
100                             mi = minute + "";
101                         }
102                         //将设置的结果保存到 etTime 中
103                         etTime.setText(time[0] + " " + ho + ":" + mi);
104                     }
105                 }, hours, minute, true);
106         tpd.setTitle("设置时间");
107         //显示时间控件
108         tpd.show();
109     }
110 });
111 //设置“保存”按钮监听器
112 btnCommit.setOnClickListener(new View.OnClickListener() {
113     @Override
114     public void onClick(View v) {
115         AlertDialog.Builder adb = new Builder(AddActivity.this);
116         //设置标题和信息
117         adb.setTitle("保存");
118         adb.setMessage("确定要保存吗？");
119         //设置按钮功能
```

```
120         adb.setPositiveButton("保存",
121                 new DialogInterface.OnClickListener() {
122                     @Override
123                     public void onClick(DialogInterface dialog,
124                             int which) {
125                         //保存备忘录信息
126                         saveNote();
127                     }
128                 });
129         adb.setNegativeButton("取消",
130                 new DialogInterface.OnClickListener() {
131                     @Override
132                     public void onClick(DialogInterface dialog,
133                             int which) {
134                         Toast.makeText(AddActivity.this, "不保存",
135                                 Toast.LENGTH_SHORT).show();
136                     }
137                 });
138         //显示对话框
139         adb.show();
140     }
141 });
142 //设置“取消”按钮监听器
143 btnCancel.setOnClickListener(new View.OnClickListener() {
144     @Override
145     public void onClick(View v) {
146         AlertDialog.Builder adb = new Builder(AddActivity.this);
147         //设置标题和消息
148         adb.setTitle("提示");
149         adb.setMessage("确定不保存吗？");
150         //设置按钮监听器
151         adb.setPositiveButton("确定",
152                 new DialogInterface.OnClickListener() {
153                     @Override
154                     public void onClick(DialogInterface dialog,
155                             int which) {
156                         //进入主界面
157                         Intent intent = new Intent();
158                         intent.setClass(AddActivity.this,
159                                 MainActivity.class);
160                         startActivity(intent);
161                     }
162                 });
163         adb.setNegativeButton("取消", null);
164         //显示对话框
165         adb.show();
166     }
```

（3）保存备忘录的函数如下所示，根据编辑模式与否，选择是添加记录或者更新记录，保存完记录之后还要设置闹钟。代码 46～56 行为设置闹钟的函数，通过 PendingIntent 将备忘录的标题和内容信息传递给 AlarmNote，在指定时间将会调用 AlarmNote，并显示标题和内容信息。

```
01  //设置保存备忘录
02  public void saveNote() {
03      //取得输入的内容
04      String name = etName.getText().toString().trim();
```

```
05        String content = etMain.getText().toString().trim();
06        String time = etTime.getText().toString().trim();
07        //内容和标题都不能为空
08        if ("".equals(name) || "".equals(content)) {
09            Toast.makeText(this, "名称和内容都不能为空", Toast.LENGTH_SHORT)
10                    .show();
11        } else {
12            if(EDIT)
13            {
14                am.saveNote(sdb, name, content, noteId, time);
15                Toast.makeText(this, "更新成功", Toast.LENGTH SHORT)
16                .show();
17            }
18            else
19            {
20                am.addNote(sdb, name, content, time);
21                Toast.makeText(this, "添加成功", Toast.LENGTH_SHORT)
22                    .show();
23            }
24            //分割日期和时间
25            String[] t = etTime.getText().toString().trim().split(" ");
26            //分割日期
27            String[] t1 = t[0].split("-");
28            //分割时间
29            String[] t2 = t[1].split(":");
30            //实例化日历
31            Calendar c2 = Calendar.getInstance();
32            //设置日历为闹钟的时间
33            c2.set(Integer.parseInt(t1[0]), Integer.parseInt(t1[1])-1,
34                    Integer.parseInt(t1[2]), Integer.parseInt(t2[0]),
35                    Integer.parseInt(t2[1]));
36            c=Calendar.getInstance();
37            //闹钟的时间应至少比现在多 10s
38            if (c.getTimeInMillis() + 1000 * 10 <= c2.getTimeInMillis()) {
39                String messageContent;
40                //当内容字数大于 20 个字时，切掉一部分以“...”代替，并存储在
                  messageContent 中
41                if (content.length() > 20) {
42                    messageContent = content.substring(0, 18) + "...";
43                } else {
44                    messageContent = content;
45                }
46                Intent intent = new Intent();
47                intent.setClass(this, AlarmNote.class);
48                //传递标题和内容信息
49                intent.putExtra("messageTitle", name);
50                intent.putExtra("messageContent", messageContent);
51                pi = PendingIntent.getBroadcast(this, 0, intent,
52                        PendingIntent.FLAG_UPDATE_CURRENT);
53                //获得闹钟服务
54                alm = (AlarmManager) getSystemService(ALARM_SERVICE);
55                //设置闹钟
56                alm.set(AlarmManager.RTC_WAKEUP, c2.getTimeInMillis(), pi);
57            }
58            Intent intent2 = new Intent();
59            intent2.setClass(this, MainActivity.class);
60            //回到主目录
61            startActivity(intent2);
62            AddActivity.this.finish();
```

```
63      }
64  }
```

（4）最后我们要为页面设置菜单选项，如 02～08 行所示。接着为菜单选项设置相应的功能，如代码 11～47 行所示。代码 50～76 行设置了 Back 按键的功能，保证当用户意外按到 Back 键时不会立即退出，而会询问用户是否保存当前文档。如代码 78～83 行所示，我们在 onDestroy()函数中关闭了数据库，这样就确保界面退出前关闭了数据库。

```
01  //新建菜单选项
02  @Override
03  public boolean onCreateOptionsMenu(Menu menu) {
04      menu.add(0, 1, 1, "关于");
05      menu.add(0, 2, 2, "设置闹铃");
06      menu.add(0, 3, 3, "退出");
07      return super.onCreateOptionsMenu(menu);
08  }
09  //为菜单选项绑定监听器
10  @Override
11  public boolean onMenuItemSelected(int featureId, MenuItem item) {
12      switch (item.getItemId()) {
13      //关于
14      case 1:
15          AlertDialog.Builder adb = new Builder(AddActivity.this);
16          adb.setTitle("关于");
17          adb.setMessage("备忘录 V1.0");
18          adb.setPositiveButton("确定", null);
19          adb.show();
20          break;
21      //设置闹铃
22      case 2:
23          Intent intent = new Intent();
24          intent.setClass(AddActivity.this, SetAlarm.class);
25          startActivity(intent);
26          break;
27      //退出
28      case 3:
29          AlertDialog.Builder adb2 = new Builder(AddActivity.this);
30          adb2.setTitle("消息");
31          adb2.setMessage("真的要退出吗？");
32          adb2.setPositiveButton("确定", new DialogInterface.OnClick
            Listener() {
33              @Override
34              public void onClick(DialogInterface dialog, int which) {
35                  //关闭列表中的所有 Activity
36                  am.exitAllProgress();
37              }
38          });
39          adb2.setNegativeButton("取消", null);
40          //显示对话框
41          adb2.show();
42          break;
43      default:
```

```
44          break;
45      }
46      return super.onMenuItemSelected(featureId, item);
47  }
48  //按键判断
49  @Override
50  public boolean onKeyDown(int keyCode, KeyEvent event) {
51      //当按键是返回键时
52      if (keyCode == KeyEvent.KEYCODE_BACK) {
53          AlertDialog.Builder adb = new Builder(AddActivity.this);
54          adb.setTitle("消息");
55          adb.setMessage("是否要保存？");
56          adb.setPositiveButton("保存", new DialogInterface.OnClick
            Listener() {
57              @Override
58              public void onClick(DialogInterface dialog, int which) {
59                  //保存备忘录
60                  saveNote();
61              }
62          });
63          adb.setNegativeButton("不保存", new DialogInterface.OnClick
            Listener() {
64              @Override
65              public void onClick(DialogInterface dialog, int which) {
66                  Intent intent2 = new Intent();
67                  intent2.setClass(AddActivity.this, MainActivity.class);
68                  //回到主页面
69                  startActivity(intent2);
70              }
71          });
72          //显示对话框
73          adb.show();
74      }
75      return super.onKeyDown(keyCode, event);
76  }
77  @Override
78  public void onDestroy()
79  {
80      super.onDestroy();
81      //关闭数据库连接
82      sdb.close();
83  }
```

6.5　闹钟设置和实现

当我们单击主页面菜单中或者备忘录添加页面菜单中的“设置铃声”选项时，页面将跳转到 SetAlarm.java，如图 6.4 所示。

下面分几个步骤讲解：

（1）铃声设置的页面设计代码如下所示，最上面使用一个 TextView 显示“音乐列表”

4 个字，下面使用 ListView 显示音乐文件列表。



图 6.4　铃声设置

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:orientation="vertical"
04      android:layout_width="match_parent"
05      android:layout_height="match_parent"
06      >
07      <!-- 标题 -->
08      <TextView
09          android:id="@+id/title"
10          android:layout_width="match_parent"
11          android:layout_height="wrap_content"
12          android:text="@string/music"
13          android:gravity="center_vertical|center_horizontal"
14          android:textSize="20px"
15          />
16      <!-- 显示 SD 卡目录下的音乐文件列表 -->
17      <ListView
18          android:id="@+id/list"
19          android:layout_width="match_parent"
20          android:layout_height="match_parent"
21          android:background="#aaaaaa"
22      />
23  </LinearLayout>
```

（2）对应的 ListView 元素的界面代码如下所示：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:orientation="horizontal"
04      android:layout_width="match_parent"
05      android:layout_height="match_parent"
06      android:background="@drawable/item"
```

```
07      >
08      <!-- 显示音乐文件名称 -->
09      <TextView
10          android:id="@+id/musicName"
11          android:layout_width="match_parent"
12          android:layout_height="30px"
13          android:gravity="center_vertical|left"
14          android:textSize="17px"
15          android:layout_marginLeft="10px"
16          android:textColor="#ff0000"
17      />
18  </LinearLayout>
```

（3）设置闹铃界面对应的程序代码如下所示，先声明需要用到的几个变量，将音乐的搜索目录设置为 SD 根目录。在初始化函数 onCreate()中执行 musicList 初始化音乐列表，在函数 musicList()中遍历 SD 根目录，搜索所有以.mp3 结尾的文件，并显示在列表中。接着为列表元素绑定监听器，如代码 40 行所示，当单击“列表元素”时将弹出对话框，询问用户是否将当前选中的音乐设置为铃声，单击“确定”按钮将会保存当前的铃声方案。

```
01  package com.guo.memorandum;          //声明包语句
02~23 行为引入相关类，这里不再列举，请阅读光盘内容
//……
24  public class SetAlarm extends Activity {
25      //显示音乐文件的列表
26      private ListView listV;
27      //列表适配器
28      private SimpleAdapter sa;
29      //音乐文件搜索路径
30      private static final String MUSIC_PATH = new String("/sdcard/");
31      @Override
32      protected void onCreate(Bundle savedInstanceState) {
33          // TODO Auto-generated method stub
34          super.onCreate(savedInstanceState);
35          setContentView(R.layout.musicmain);
36          listV = (ListView) findViewById(R.id.list);
37          //显示音乐文件
38          musicList();
39          //设置按键监听器
40          listV.setOnItemClickListener(new OnItemClickListener() {
41              @SuppressWarnings("unchecked")
42              @Override
43              public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
44                      long arg3) {
45                  Map<String, String> map = (Map<String, String>)
                    arg0.getItemAtPosition(arg2);
46                  //取得音乐文件名称
47                  final String name = map.get("musicName");
48                  //创建对话框
49                  AlertDialog.Builder adb = new Builder(SetAlarm.this);
50                  adb.setTitle("提示消息");
51                  adb.setMessage("确定要将 "+name+" 设置为默认闹铃声吗？");
52                  adb.setPositiveButton("确定", new DialogInterface
                    .OnClickListener() {
53                      @Override
54                      public void onClick(DialogInterface dialog,int which) {
55                          Uri uri = Uri.parse(MUSIC_PATH + name);
56                          //设置闹铃的路径
```

```
57                      ActivityManager.setUri(uri);
58                      Toast.makeText(SetAlarm.this, "设置成功",Toast
                        .LENGTH_SHORT).show();
59                      //关闭当前页面
60                      finish();
61                   }
62                });
63                adb.setNegativeButton("取消",null);
64                //显示对话框
65                adb.show();
66             }
67          });
68       }
69       //显示音乐文件列表
70       public void musicList() {
71          //取得需要遍历的文件目录
72          File home = new File(MUSIC_PATH);
73          List<Map<String, String>> list = new ArrayList<Map<String,
            String>>();
74          //遍历文件目录
75          if (home.listFiles(new MusicFilter()).length > 0) {
76             for (File file : home.listFiles(new MusicFilter())) {
77                Map<String, String> map = new HashMap<String, String>();
78                map.put("musicName", file.getName());
79                list.add(map);
80             }
81             sa = new SimpleAdapter(SetAlarm.this, list, R.layout
               .musicitems,
82                   new String[] { "musicName" }, new int[] { R.id.musicName });
83             listV.setAdapter(sa);
84          }
85       }
86    }
87    //过滤所有不是以.mp3结尾的文件
88    class MusicFilter implements FilenameFilter {
89       public boolean accept(File dir, String name) {
90          return (name.endsWith(".mp3"));
91       }
92    }
```

（4）当闹钟触发时，将会发送一个广播并被接收器 AlarmNote 接收，AlarmNote 获取传递过来的 Intent 中绑定的标题和内容信息，将其绑定到一个新的 Intent 中，并启动一个新的界面 Alarm。

```
01    package com.guo.memorandum;              //声明包语句
02~05 行为引入相关类，这里不再列举，请阅读光盘内容
//……
06    public class AlarmNote extends BroadcastReceiver {
07       @Override
08       public void onReceive(Context context, Intent intent) {
09          // TODO Auto-generated method stub
10          //获得标题
11          String messageTitle=intent.getStringExtra("messageTitle");
12          //获得内容
13          String messageContent=intent.getStringExtra("messageContent");
14          Intent in=new Intent();
15          in.setClass(context, Alarm.class);
16          in.putExtra("messageTitle", messageTitle);
```

```
17          in.putExtra("messageContent", messageContent);
18          in.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
19          //调用 Alarm
20          context.startActivity(in);
21      }
22  }
```

（5）在 Alarm 中主要实现播放闹铃，并显示对话框让用户可以关掉闹铃。如下所示，代码 35～43 行用于显示备忘录的标题和内容。

```
01  package com.guo.memorandum;                //声明包语句
02~14 行为引入相关类，这里不再列举，请阅读光盘内容
//……
15  public class Alarm extends Activity {
16      //媒体播放器
17      private MediaPlayer mMediaPlayer;
18      @Override
19      protected void onCreate(Bundle savedInstanceState) {
20          // TODO Auto-generated method stub
21          super.onCreate(savedInstanceState);
22          setContentView(R.layout.alarm);
23          try {
24              //播放指定的音乐
25              mMediaPlayer=MediaPlayer.create(Alarm.this,ActivityManager
                .getUri());
26              //设置播放的音量
27              mMediaPlayer.setVolume(300, 350);
28              //设置循环
29              mMediaPlayer.setLooping(true);
30          } catch (Exception e) {
31              Toast.makeText(Alarm.this,"音乐文件播放异常",Toast
                .LENGTH_SHORT);
32          }
33          //开始播放
34          mMediaPlayer.start();
35          Intent intent=getIntent();
36          //获得标题
37          String messageTitle=intent.getStringExtra("messageTitle");
38          //获得内容
39          String messageContent=intent.getStringExtra("messageContent");
40          //新建对话框
41          AlertDialog.Builder adb=new Builder(Alarm.this);
42          adb.setTitle(messageTitle);
43          adb.setMessage(messageContent);
44          adb.setPositiveButton("确定", new OnClickListener() {
45              @Override
46              public void onClick(DialogInterface dialog, int which) {
47                  //关闭媒体播放器
48                  mMediaPlayer.stop();
49                  mMediaPlayer.release();
50                  finish();
51              }
52          });
53          //显示对话框
54          adb.show();
55      }
```

6.6　公共类的实现

最后我们来讲解一下公共类的实现，新建 ActivityManager.java，代码如下所示。这个类主要用来实现整个程序需要共同使用的一些函数，代码 15 行声明了一个静态变量 instance 用于存储 ActivityManager 的实例，每次对这个类进行实例化，得到的都是同一个实例。

getUri 和 setUri 分别用于获取和设置铃声的地址，exitAllProcess()函数用于在程序退出时调用，以关闭所有界面，addNote 和 saveNote 分别用于添加备忘录和保存备忘录，returnTime 用于以特定格式返回当前的时间和日期。

```
01  package com.guo.memorandum;             //声明包语句
02~11 行为引入相关类，这里不再列举，请阅读光盘内容
//……
12  //活动管理器
13  public class ActivityManager {
14      //用静态变量存储实例
15      private static ActivityManager instance;
16      private List<Activity> list;
17      //默认铃声的地址
18      private static Uri uri=Uri.parse("/sdcard/demo.mp3");
19
20      public static ActivityManager getInstance() {
21          if (instance == null)
22              instance = new ActivityManager();
23          return instance;
24      }
25      //添加 Activity 进列表
26      public void addActivity(Activity av) {
27          //如果列表为空则新建列表
28          if(list==null)
29              list=new ArrayList<Activity>();
30          if (av != null) {
31              list.add(av);
32          }
33      }
34      //获得铃声的路径
35      public static Uri getUri() {
36          return uri;
37      }
38      //设置铃声
39      public static void setUri(Uri uri) {
40          ActivityManager.uri = uri;
41      }
42      //退出所有程序
43      public void exitAllProgress() {
44          for (int i = 0; i < list.size(); i++) {
45              Activity av = list.get(i);
46              av.finish();
47          }
48      }
49      //更新文件
50      public void saveNote(SQLiteDatabase sdb,String name,String content,
```

```
        String noteId,String time){
51          ContentValues cv=new ContentValues();
52          cv.put("noteName", name);
53          cv.put("noteContent", content);
54          cv.put("noteTime", time);
55          sdb.update("note", cv, "noteId=?", new String[]{noteId});
56      }
57      //添加文件
58      public void addNote(SQLiteDatabase sdb,String name,String content,
        String time){
59          ContentValues cv=new ContentValues();
60          cv.put("noteName", name);
61          cv.put("noteContent", content);
62          cv.put("noteTime", time);
63          sdb.insert("note", null, cv);
64      }
65      //返回当前的时间
66      public String returnTime(){
67          Date d=new Date();
68          SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd HH:
            mm:ss");
69          String time=sdf.format(d);
70          return time;
71      }
```

至此，备忘录的全部内容已分析完，需要注意的一点是，由于本程序闹钟的地址保存在静态变量中，因此每次重启或者程序完全退出后铃声地址都会被重置。改进的办法也很简单，可以将铃声的地址使用 sharedPreferences 保存，这样就不会受到重启等其他因素的影响了。

6.7　知 识 拓 展

本章在设置闹铃的时候用到了 AlarmManager 类，下面我们来进一步了解一下这个类的一些特性。这个类主要用在当程序不再运行，而此时你仍然需要你的应用去执行一些操作（如闹钟）的情况，也就是说其他大多数情况可以不适用这个类，而用 Handler 等代替。

AlarmManager 包含的主要方法有如下：

```
01  // 取消已经注册的与参数匹配的定时器
02  void   cancel(PendingIntent operation)
03  //注册一个新的延迟定时器
04  void   set(int type, long triggerAtTime, PendingIntent operation)
05  //注册一个重复类型的定时器
06  void   setRepeating(int type, long triggerAtTime, long interval, Pending
    Intent operation)
07  //注册一个非精密的重复类型的定时器
08  void setInexactRepeating (int type, long triggerAtTime, long interval,
    PendingIntent operation)
09  //设置时区
10  void   setTimeZone(String timeZone)
```

其中的 type，也就是定时器的类型可选择的范围如下所示：

```
01  //当系统进入睡眠状态时，这种类型的闹铃不会唤醒系统，直到系统下次被唤醒才传递它，
```

```
02 //该闹铃所用的时间是相对时间，是从系统启动后开始计时的，包括睡眠时间，
03 //可以通过调用 SystemClock.elapsedRealtime()获得。系统值是 3 (0x00000003)
04 public  static  final  int  ELAPSED_REALTIME
05
06  //能唤醒系统，用法同 ELAPSED_REALTIME，系统值是 2 (0x00000002)
07 public  static  final  int  ELAPSED REALTIME WAKEUP
08
09 //当系统进入睡眠状态时，这种类型的闹铃不会唤醒系统的，
10 //直到系统下次被唤醒才传递它，该闹铃所用的时间是绝对时间，所用时间是 UTC 时间，
11 ///可以通过调用 System.currentTimeMillis()获得。系统值是 1 (0x00000001)
12 public  static  final  int  RTC
13
14 //能唤醒系统，用法同 RTC 类型，系统值为 0 (0x00000000)
15 public  static  final  int  RTC WAKEUP
16
17 //能唤醒系统，它是一种关机闹铃，就是说设备在关机状态下也可以唤醒系统，
18 //所以我们把它称之为关机闹铃。使用方法同 RTC 类型，系统值为 4(0x00000004)
```

AlarmManager 使用时需要注意一些细节，比如当你执行了 repeating AlarmManager，则这个 AlarmManager 将会一直在后台运行，除非你在“应用管理”中“强行停止”掉这个 AlarmManager。此外，如果某个 AlarmManager 已经启动，若程序再次就调用它，只要 PendingIntent 是一样的，则之前的 AlarmManager 将会被释放掉，被当前的 AlarmManager 替代。

AlarmManager 总共有 3 种使用方式，都是通过 PendingIntent，如下所示：

```
1  //启动一个 Activity
2  getActivity(Context, int, Intent, int)
3  //触发一个接收器，即发送一个广播
4  getBroadcast(Context, int, Intent, int)
5  //调用一个服务
6  getService(Context, int, Intent, int)
```

6.8　本 章 小 结

本章讲述了如何开发一个备忘录程序，使用 sqlite 存储数据，并利用 AlarmManager 设置定时闹钟。通过本章的学习，读者应该更加熟练 sqlite 这种数据存储方式的使用，同时要学会在一些必要的场合运用 AlarmManager 来完成特殊的功能。

第 7 章　短信收发工具

Android 作为一个手机操作系统，自然少不了收发短信的功能，本章将要介绍的就是如何实现短信的收发功能。

7.1　显示手机所有信息

程序的开始我们要获取手机中所有的短信息，如图 7.1 所示为显示手机所有信息的界面。可以看出这个界面主要由两部分组成，一个是最上方的“新建信息”按钮，另一个就是用于显示信息的 List 部分。

图 7.1　显示手机所有信息的界面

7.1.1　新建 main.xml

根据这个界面的设计分析，我们在 res/layout 下新建 main.xml，代码如下所示，采用 LinearLayout 布局，前面放置一个 Button 用于新建信息，主体部分放一个 ListView 用于显

示短信内容。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
03     android:layout_width="match_parent"
04     android:layout_height="fill_parent"
05     android:background="#ffffff"
06     android:orientation="vertical" >
07  <!--“新建信息”按钮 -->
08     <Button
09        android:id="@+id/newsms"
10        android:layout width="fill parent"
11        android:layout height="45dp"
12        android:background="@drawable/button"
13        android:text="@string/newsms"
14        android:textColor="#65516A"
15        android:textSize="25dp" />
16  <!--信息列表 -->
17     <ListView
18        android:id="@+id/msg list"
19        android:layout width="fill parent"
20        android:layout height="wrap content"
21        android:background="#00000000"
22        android:scrollbars="vertical"
23        android:text="@string/newsms" >
24     </ListView>
25  </LinearLayout>
```

7.1.2　设置布局

在 res/layout 下新建 showlist.xml 为每行元素设定布局，如下代码所示。每一行分为左边和右边两部分，左边显示头像，右边显示文字，而文字包括手机号码、已发送和已接收的标志、信息内容、发送日期等。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout xmlns:android="http://schemas.android.com/
    apk/res/android"
03     android:id="@+id/rootLayout"
04     android:layout width="match parent"
05     android:layout height="match parent"
06     android:background="#00000000"
07     android:orientation="horizontal" >
08     <!-- 显示头像 -->
09     <ImageView
10        android:id="@+id/imag"
11        android:layout_width="55dp"
12        android:layout_height="60dp"
13        android:background="@drawable/people" />
14     <!-- 文本区域 -->
15     <RelativeLayout
16        android:layout width="wrap content"
17        android:layout height="55dp"
18        android:layout marginLeft="10dp"
19        android:layout toRightOf="@+id/imag" >
20        <!-- 手机号码 -->
21        <TextView
```

```
22              android:id="@+id/listnum"
23              android:layout width="wrap content"
24              android:layout height="25dp"
25              android:gravity="center vertical"
26              android:textColor="#000000" />
27          <!-- 显示"收"或者"发"，分别表示发件箱和收件箱的邮件 -->
28          <TextView
29              android:id="@+id/type"
30              android:layout_width="wrap_content"
31              android:layout_height="25dp"
32              android:layout_above="@+id/relativeLayout1"
33              android:layout alignParentRight="true"
34              android:textColor="#006633"
35              android:textStyle="bold"
36              android:textSize="22dp"
37              android:layout marginRight="20dp"/>
38          <!-- 消息部分 -->
39          <RelativeLayout
40              android:id="@+id/relativeLayout1"
41              android:layout_width="match_parent"
42              android:layout_height="wrap_content"
43              android:layout_below="@+id/listnum" >
44              <!-- 消息 -->
45              <TextView
46                  android:id="@+id/listmsg"
47                  android:layout width="wrap content"
48                  android:layout height="match parent"
49                  android:gravity="center vertical"
50                  android:lines="1"
51                  android:maxLength="10"
52                  android:textColor="#000000" />
53              <!-- 日期 -->
54              <TextView
55                  android:id="@+id/listtime"
56                  android:layout width="wrap content"
57                  android:layout height="match parent"
58                  android:layout alignParentRight="true"
59                  android:layout marginRight="10dp"
60                  android:gravity="center vertical"
61                  android:textColor="#000000" />
62          </RelativeLayout>
63      </RelativeLayout>
64      <!-- 分割条 -->
65      <TextView
66          android:layout_width="fill_parent"
67          android:layout_height="1dp"
68          android:layout_alignBottom="@+id/imag"
69          android:layout_alignParentLeft="true"
70          android:background="#bbbbbb" />
71  </RelativeLayout>
```

7.1.3　新建文件 MsgListActvity.java

在包 guo.supermario.sms 目录下新建文件 MsgListActvity.java，如下所示。在 onCreate() 函数中对界面元素进行初始化，为“新建信息”按钮绑定监听器，当单击该按钮，界面将跳转到发送信息界面，如代码 039～047 行所示。

接着新建简单适配器 adapter，如代码 049～052 行所示，通过函数 getSmsInPhone()获

得手机中所有短信息，并将信息显示到界面的对应位置。函数 getSmsInPhone()中通过查询 uri 地址 content://sms/获取短信息对应的列——address、person、body、date、type，接着对日期进行格式化处理，把表示短信息类型的数字格式转化成文字格式，并存储到 map 中。每一条短信息对应一个 map，将所有的 map 添加到一个 List 中返回。

最后为列表元素添加按钮监听器，当用户单击列表元素时，界面将跳转到回复界面。

```
001 package guo.supermario.sms;            //声明包语句
002~023 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
024 //显示收到的信息列表
025 public class MsgListActivity extends Activity {
026     //“新建信息”按钮
027     private Button sendnmsg;
028     //信息列表
029     private ListView msglist;
030     //初始化函数
031     public void onCreate(Bundle savedInstanceState) {
032         super.onCreate(savedInstanceState);
033         //去除标题
034         requestWindowFeature(Window.FEATURE_NO_TITLE);
035         setContentView(R.layout.main);
036         //初始化界面元素
037         sendnmsg = (Button) findViewById(R.id.newsms);
038         msglist = (ListView) findViewById(R.id.msg_list);
039         sendnmsg.setOnClickListener(new OnClickListener() {
040             public void onClick(View v) {
041                 //启动“新建信息”界面
042                 Intent intent = new Intent(MsgListActivity.this,
043                         SendSmsActivity.class);
044                 MsgListActivity.this.startActivity(intent);
045                 MsgListActivity.this.finish();
046             }
047         });
048         //新建适配器
049         SimpleAdapter adapter = new SimpleAdapter(MsgListActivity.this,
050                 getSmsInPhone(), R.layout.showlist, new String[]
                { "imag",
051                         "listnum", "listmsg", "listtime" ,"listtype"}, new
                         int[] {
052                                 R.id.imag, R.id.listnum, R.id.listmsg,
                                R.id.listtime ,R.id.type});
053         //通知数据更改
054         adapter.notifyDataSetChanged();
055         //设置适配器
056         msglist.setAdapter(adapter);
057         //设置项目按钮监听器
058         msglist.setOnItemClickListener(new OnItemClickListener() {
059             public void onItemClick(AdapterView<?> arg0, View arg1,
060                     int position, long arg3) {
061                 //启动回复信息界面
062                 Intent intent = new Intent(MsgListActivity.this,
                 Replymsg.class);
063                 //取得对应类表项目的元素
064                 Map<String, Object> getlist = getSmsInPhone().get
                (position);
```

```
065                     //将数据绑定到 intent 中进行传递
066                     intent.putExtra("listnum", (String) getlist.get
                        ("listnum"));
067                     intent.putExtra("listtime", (String) getlist.get
                        ("listtime"));
068                     intent.putExtra("listmsg", (String) getlist.get
                       ("listmsg"));
069                     intent.putExtra("listtype", (String)getlist.
                        get("listtype"));
070                     MsgListActivity.this.startActivity(intent);
071                     MsgListActivity.this.finish();
072                 }
073             });
074
075         }
076         //获取手机中的所有信息
077         public List<Map<String, Object>> getSmsInPhone() {
078             //所有信息对应的 uri 地址
079             final String SMS URI ALL = "content://sms/";
080             List<Map<String, Object>> contents = new ArrayList<Map<String,
                Object>>();
081             try {
082                 Uri uri = Uri.parse(SMS URI ALL);
083                 //所要查询的列
084                 String[] projection = new String[] { " id", "address",
                   "person",
085                         "body", "date", "type" };
086                 //查询
087                 Cursor cur = getContentResolver().query(uri, projection,
088                 null,null, "date desc"); //获取手机内部短信
089                 //将查询结果保存到列表中
090                 while (cur.moveToNext()) {
091                     Map<String, Object> map = new HashMap<String, Object>();
092                     //取得列的索引
093                     int index_Address = cur.getColumnIndex("address");
094                     int index Person = cur.getColumnIndex("person");
095                     int index Body = cur.getColumnIndex("body");
096                     int index Date = cur.getColumnIndex("date");
097                     int index Type = cur.getColumnIndex("type");
098                     //取得短信发送目标的电话
099                     String strAddress = cur.getString(index Address);
100                     //取得索引
101                     cur.getInt(index_Person);
102                     //取得信息的内容
103                     String strbody = cur.getString(index Body);
104                     //取得日期
105                     long longDate = cur.getLong(index Date);
106                     //取得短信的类型，1 为已接收，2 为已发送
107                     int type= cur.getInt(index Type);
108                     //格式化日期
109                     SimpleDateFormat dateFormat = new SimpleDateFormat(
110                             "yyyy-MM-dd hh:mm:ss");
111                     Date d = new Date(longDate);
112                     String strDate = dateFormat.format(d);
113                     //将数据存储到 map 中
114                     map.put("listnum", strAddress);
115                     map.put("listmsg", strbody);
116                     map.put("listtime", strDate);
```

```
117                 if(type == 1)
118                 {
119                     map.put("listtype", "收");
120                 }else{
121                     map.put("listtype", "发");
122                 }
123                 //将 map 添加到列表中
124                 contents.add(map);
125             }
126             if (!cur.isClosed()) {
127                 cur.close();
128                 cur = null;
129             }
130         } catch (SQLiteException ex) {
131             Log.d("SQLiteException in getSmsInPhone", ex.getMessage());
132         }
133             //返回信息列表
134             return contents;
135         }
136 }
```

7.2　新建信息

7.2.1　界面设计

新建信息界面如图 7.2 所示，主要由收件人号码、发送内容和一个“发送”按钮组成。界面代码如下所示，最上面放置一个 EditText 用于给用户输入收件人号码，左边放置一个 ImageView 作为 logo。界面的中间放置一个 EditText 用于输入发送内容，最下面一个是“发送”按钮。



图 7.2　新建信息界面

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout_width="match_parent"
04      android:layout_height="fill_parent"
05      android:background="#555555"
```

```
06      android:orientation="vertical" >
07      <LinearLayout
08          android:layout_width="fill_parent"
09          android:layout_height="wrap_content"
10          android:orientation="horizontal" >
11          <!-- logo 图片 -->
12          <ImageView
13              android:id="@+id/contact"
14              android:layout_width="45dp"
15              android:layout_height="45dp"
16              android:layout_marginLeft="20dp"
17              android:layout_marginTop="10dp"
18              android:background="@drawable/contact" />
19          <!-- 电话号码输入框 -->
20          <EditText
21              android:id="@+id/ednum"
22              android:layout_width="fill_parent"
23              android:layout_height="45dp"
24              android:layout_marginRight="20dp"
25              android:layout_marginTop="10dp"
26              android:background="#DEEBFE"
27              android:hint="@string/sendnum"
28              android:numeric="integer"
29              android:singleLine="true" />
30      </LinearLayout>
31      <!-- 发送内容 -->
32      <EditText
33          android:id="@+id/edbody"
34          style="@android:style/Theme.Translucent"
35          android:layout_width="fill_parent"
36          android:layout_height="wrap_content"
37          android:layout_margin="20dp"
38          android:layout_weight="0.69"
39          android:background="#E0E0FC"
40          android:gravity="left"
41          android:hint="@string/body"
42          android:maxLines="11"
43          android:minLines="11"
44          android:scrollbars="vertical" />
45      <!--“发送”按钮 -->
46      <Button
47          android:id="@+id/send"
48          android:layout_width="wrap_content"
49          android:layout_height="wrap_content"
50          android:layout_gravity="center"
51          android:layout_marginBottom="20dp"
52          android:background="@drawable/send"
53          android:text="发送" />
54  </LinearLayout>
```

7.2.2　实现发送信息功能

新建 SendSmsActivity.java 用于实现发送信息功能，代码如下所示，在 onCreate()函数中初始化界面元素，注册两个广播接收器：一个是 send_sms，用于接收当用户发送信息之后，系统反馈的信息发送的状况；另一个是 delievered_sms，用于接收当短信经过供应商发送出去之后，从收件人反馈回来的信息。注册广播采用 registerReceiver()函数，同时记得要

在 onDestroy()函数中调用函数 unregisterReceiver()来注销广播接收器。

当单击“发送”按钮时，将调用类 SendMessage 的方法 sendmsg 发送信息，同时将发送的短信存储到系统的发件箱中，最后调用函数 back()，关闭当前界面，返回信息列表界面。

```
001 package guo.supermario.sms;            //声明包语句
002~016 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
017 //发送信息界面
018 public class SendSmsActivity extends Activity {
019     //收件人号码
020     EditText ednum;
021     //短信内容
022     EditText edbody;
023     //设置“发送”按钮
024     Button send;
025     //发送状态广播
026     BroadcastReceiver send_sms;
027     //对方接收状态广播
028     BroadcastReceiver delivered_sms;
029     String SENT_SMS_ACTION = "SENT_SMS_ACTION";
030     String DELIVERED_SMS_ACTION = "DELIVERED_SMS_ACTION";
031     String getnum;
032     String getmsg;
033     @Override
034     public void onCreate(Bundle savedInstanceState) {
035         super.onCreate(savedInstanceState);
036         setContentView(R.layout.sendmsg);
037         //界面初始化
038         ednum = (EditText) findViewById(R.id.ednum);
039         edbody = (EditText) findViewById(R.id.edbody);
040         send = (Button) findViewById(R.id.send);
041         //设置“发送”按钮监听器
042         send.setOnClickListener(new OnClickListener() {
043             public void onClick(View v) {
044                 //取得输入的号码
045                 getnum = ednum.getText().toString();
046                 //取得输入的内容
047                 getmsg = edbody.getText().toString();
048                 //实例化发送信息类
049                 SendMassage sendmsg = new SendMassage(SendSmsActivity.
                    this,
050                         getmsg, getnum);
051                 //发送信息
052                 sendmsg.sendmsg(send_sms,delivered_sms);
053                 //将信息存储到数据库“已发送”中
054                 ContentValues values=new ContentValues();
055                 //发送时间
056                 values.put("date", System.currentTimeMillis());
057                 //阅读状态
058                 values.put("read", 0);
059                 //1 为收，2 为发
060                 values.put("type", 2);
061                 //送达号码
062                 values.put("address", getnum);
```

```
063                 //发送内容
064                 values.put("body", getmsg);
065                 getContentResolver().insert(Uri.parse("content://sms/
                    sent"), values);
066                 edbody.setText("");
067                 //返回信息列表界面
068                 back();
069
070             }
071         });
072         send_sms=new BroadcastReceiver() {
073             @Override
074             public void onReceive(Context  context, Intent  intent) {
075                 switch (getResultCode()) {
076                 //发送成功
077                 case Activity.RESULT_OK:
078                     Toast.makeText(_context,
079                             "短信发送成功！", Toast.LENGTH_SHORT)
080                             .show();
081                     break;
082                     //通用错误
083                 case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
084                     Toast.makeText(_context,
085                             "SMS generic failure actions", Toast.
                            LENGTH_SHORT)
086                             .show();
087                     break;
088                     //无线电被关闭
089                 case SmsManager.RESULT_ERROR_RADIO_OFF:
090                     Toast
091                             .makeText(_context,
092                                     "SMS radio off failure actions",
093                                     Toast.LENGTH_SHORT).show();
094                     break;
095                     //未提供 pdu(协议描述单元 protocol description unit)
096                 case SmsManager.RESULT_ERROR_NULL_PDU:
097                     Toast.makeText(_context,
098                             "SMS null PDU failure actions", Toast.LENGTH_
                            SHORT)
099                             .show();
100                     break;
101                 }
102             }
103         };
104         delivered_sms=new BroadcastReceiver() {
105             @Override
106             public void onReceive(Context  context, Intent  intent) {
107                 Toast.makeText(_context, "短信已被接收！",
108                         Toast.LENGTH_SHORT).show();
109             }
110         };
111         //注册广播接收器
112         registerReceiver(send_sms, new IntentFilter(SENT_SMS_ACTION));
113         registerReceiver(delivered_sms, new IntentFilter(DELIVERED_
            SMS_ACTION));
114     }
115     //返回信息列表
116     private void back() {
117         Intent intent = new Intent(SendSmsActivity.this,
            MsgListActivity.class);
```

```
118         SendSmsActivity.this.startActivity(intent);
119         SendSmsActivity.this.finish();
120
121     }
122     @Override
123     public void onDestroy()
124     {
125         //注销广播接收器
126         unregisterReceiver(send_sms);
127         unregisterReceiver(delivered_sms);
128         super.onDestroy();
129     }
130 }
```

7.2.3　发送信息

发送信息类 SendMessage 代码如下所示，在构造函数中获得传递过来的上下文变量、收件人号码、短信内容。在发送短信函数 sendmsg()中新建 PendingIntent 用于发送广播，接着实例化短信管理类 SmsManager，将短信内容进行拆分发送。

```
01  package guo.supermario.sms;              //声明包语句
02~10 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
11  public class SendMassage {
12      Context context;
13      //发送的内容
14      String msg;
15      //收件人的号码
16      String mobile;
17      //构造函数
18      public SendMassage(Context context,String msg,String mobile){
19          this.context = context;
20          this.msg = msg;
21          this.mobile = mobile;
22      }
23      //发送信息
24      public void sendmsg(BroadcastReceiver send sms,BroadcastReceiver
        delivered sms){
25          String SENT SMS ACTION = "SENT SMS ACTION";
26          String DELIVERED SMS ACTION = "DELIVERED SMS ACTION";
27
28          //新建 PendingIntent 用于调用指定广播接收器
29          Intent sentIntent = new Intent(SENT_SMS_ACTION);
30          PendingIntent sentPI = PendingIntent.getBroadcast(context, 0,
            sentIntent,
31                  0);
32          //新建 PendingIntent 用于调用指定广播接收器
33          Intent deliverIntent = new Intent(DELIVERED SMS ACTION);
34          PendingIntent deliverPI = PendingIntent.getBroadcast(context, 0,
35                  deliverIntent, 0);
36          //实例化短信管理类
37          SmsManager smsManager = SmsManager.getDefault();
38          //拆分短信
39          ArrayList<String> texts = smsManager.divideMessage(msg);
40          for(String text : texts){
```

```
41              //分条发送短信
42              smsManager.sendTextMessage(mobile, null, text, sentPI,
                deliverPI);
43          }
44      }
45  }
```

7.3　回复信息

回复信息其实跟新建信息差别不大，只是界面略有不同，如图 7.3 所示，最上面放置的是已收到短信的号码和发送时间，中间放置短信内容，最下面分别放置回复内容编辑框和“回复”按钮。界面实现代码如下所示，采用 LinearLayout 进行布局，采用 TextView 来显示收到短信的详细信息，用一个 EditText 让用户进行短信输入，最下面一个 Button 用于发送信息。



图 7.3　回复信息界面

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
03      android:layout_width="match_parent"
04      android:layout_height="match_parent"
05      android:background="#eeeeee"
06      android:orientation="vertical" >
07      <LinearLayout
08          android:layout_width="fill_parent"
09          android:layout_height="wrap_content"
10          android:layout_margin="5dp"
11          android:background="#50000000"
12          android:orientation="vertical" >
13          <!-- 收件人号码 -->
14          <TextView
15              android:id="@+id/renum"
16              android:layout_width="fill_parent"
17              android:layout_height="25dp"
18              android:textColor="#000000"
```

```
19              android:textSize="20dp" />
20          <!-- 对方短信发送时间-->
21          <TextView
22              android:id="@+id/retime"
23              android:layout_width="fill_parent"
24              android:layout_height="25dp"
25              android:gravity="right"
26              android:textColor="#000000" />
27      </LinearLayout>
28      <!-- 对方短信内容 -->
29      <TextView
30          android:id="@+id/recontent"
31          android:layout width="fill parent"
32          android:layout height="0dp"
33          android:layout margin="5dp"
34          android:layout weight="3"
35          android:textColor="#000000"
36          android:background="#F5F5DC"
37          android:textSize="25dp"
38          android:autoLink="email|phone|web" />
39      <!-- 欲答复的内容 -->
40      <EditText
41          android:id="@+id/replybody"
42          android:layout width="fill parent"
43          android:layout height="wrap content"
44          android:layout margin="5dp"
45          android:gravity="left"
46          android:hint="回复内容"
47          android:maxLines="4"
48          android:minLines="4" />
49      <!–“回复”按钮 -->
50      <Button
51          android:id="@+id/reply"
52          android:layout width="fill parent"
53          android:layout height="50dp"
54          android:background="@drawable/button"
55          android:gravity="center"
56          android:text="回复" />
57  </LinearLayout>
```

（2）在程序的开始需要先初始化界面元素，如代码 043～047 行所示，并调用函数 getdata()获得上一个 Activity 传递过来的数据，填充到界面指定位置，包括收件人号码、短信内容、短信发送时间等。

当用户单击“回复”按钮时，将调用函数发送信息，并将信息存储到“发件箱”中并返回短信列表界面。同样，因为在初始化的时候注册了广播，因此需要在 onDestroy()函数中注销广播。

```
001 package guo.supermario.sms;                    //声明包语句
002~019 行为引入相关类，这里不再列举，请阅读光盘内容。
//
……
020 public class Replymsg extends Activity {
021     //显示收件人号码界面
022     private TextView phonnum;
023     //显示对方短信发送时间
024     private TextView phontime;
025     //显示对方短信内容
```

```
026     private TextView phonmsg;
027     //用于编辑发送内容
028     private EditText sendcontent;
029     // “回复” 按钮
030     private Button rereplybt;
031     private Intent intent;
032     //接收器
033     BroadcastReceiver send_sms;
034     BroadcastReceiver delivered_sms;
035     String SENT_SMS_ACTION = "SENT_SMS_ACTION";
036     String DELIVERED_SMS_ACTION = "DELIVERED_SMS_ACTION";
037     //初始化函数
038     @Override
039     public void onCreate(Bundle savedInstanceState) {
040         super.onCreate(savedInstanceState);
041         setContentView(R.layout.replysms);
042         //界面元素的初始化
043         phonnum = (TextView) findViewById(R.id.renum);
044         phontime = (TextView) findViewById(R.id.retime);
045         phonmsg = (TextView) findViewById(R.id.recontent);
046         sendcontent = (EditText) findViewById(R.id.replybody);
047         rereplybt = (Button) findViewById(R.id.reply);
048         send_sms=new BroadcastReceiver() {
049             @Override
050             public void onReceive(Context _context, Intent _intent) {
051                 switch (getResultCode()) {
052                 case Activity.RESULT_OK:
053                     Toast.makeText(_context,
054                             "短信发送成功!", Toast.LENGTH_SHORT)
055                             .show();
056                     break;
057                     //通用错误
058                 case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
059                     Toast.makeText(_context,
060                             "SMS generic failure actions", Toast.
                            LENGTH_SHORT)
061                             .show();
062                     break;
063                     //无线电被关闭
064                 case SmsManager.RESULT_ERROR_RADIO_OFF:
065                     Toast
066                             .makeText(_context,
067                                     "SMS radio off failure actions",
068                                     Toast.LENGTH_SHORT).show();
069                     break;
070                     //未提供 pdu(协议描述单元 protocol description unit)
071                 case SmsManager.RESULT_ERROR_NULL_PDU:
072                     Toast.makeText(_context,
073                             "SMS null PDU failure actions", Toast.
                            LENGTH_SHORT)
074                             .show();
075                     break;
076                 }
077             }
078         };
079         delivered_sms=new BroadcastReceiver() {
080             @Override
081             public void onReceive(Context _context, Intent _intent) {
082                 Toast.makeText(_context, "短信已被接收! ",
083                         Toast.LENGTH_SHORT).show();
```

```
084             }
085         };
086         //注册广播
087         registerReceiver(send_sms, new IntentFilter(SENT_SMS_ACTION));
088         registerReceiver(delivered_sms, new IntentFilter
            (DELIVERED_SMS_ACTION));
089         //获取信息
090         getdata();
091         //回复
092         send();
093     }
094     //获取信息
095     private void getdata() {
096         intent = getIntent();
097         //获得传递过来的信息
098         String listnum = intent.getStringExtra("listnum");
099         String listtime = intent.getStringExtra("listtime");
100         String listmsg = intent.getStringExtra("listmsg");
101         //初始化界面文本
102         phonnum.setText(listnum);
103         phontime.setText(listtime);
104         phonmsg.setText(listmsg);
105     }
106     //按键监听器
107     @Override
108     public boolean onKeyDown(int keyCode, KeyEvent event) {
109         //TODO Auto-generated method stub
110         if (keyCode == KeyEvent.KEYCODE_BACK) {
111             back();
112         }
113         return super.onKeyDown(keyCode, event);
114     }
115     //返回信息显示列表
116     private void back() {
117         Intent intentb = new Intent(Replymsg.this, MsgListActivity.
            class);
118         Replymsg.this.startActivity(intentb);
119         Replymsg.this.finish();
120     }
121     //回复短信
122     private void send() {
123         rereplybt.setOnClickListener(new OnClickListener() {
124             String getnum;
125             String getmsg;
126             public void onClick(View v) {
127                 //获得发送号码
128                 getnum = phonnum.getText().toString();
129                 //获得发送内容
130                 getmsg = sendcontent.getText().toString();
131                 //实例化发送信息类
132                 SendMassage sendmsg = new SendMassage(Replymsg.this,
                    getmsg,getnum);
133                 //发送信息
134                 sendmsg.sendmsg(send_sms,delivered_sms);
135                 ContentValues values=new ContentValues();
136                 //发送时间
137                 values.put("date", System.currentTimeMillis());
138                 //阅读状态
139                 values.put("read", 0);
```

```
140                 //1 为收，2 为发
141                 values.put("type", 2);
142                 //送达号码
143                 values.put("address", getnum);
144                 //发送内容
145                 values.put("body", getmsg);
146                 getContentResolver().insert(Uri.parse("content://sms/
                    sent"), values);
147                 sendcontent.setText("");
148                 //返回信息显示界面
149                 back();
150             }
151         });
152     }
153     //程序销毁时调用
154     @Override
155     public void onDestroy()
156     {
157         //注销函数
158         unregisterReceiver(send_sms);
159         unregisterReceiver(delivered_sms);
160         super.onDestroy();
161     }
162 }
```

7.4　知 识 拓 展

我们使用 Android 系统发送短信的时候是有字数限制的，通常我们调用 divideMessage 函数把字符串按照限制来分割成可以发的消息。那么 Android 是如何对字符串进行分割的呢？

在 PDU Mode 中，可以采用 3 种编码方式来对发送的内容进行编码，它们是 7-bit、8-bit 和 UCS2 编码。7-bit 编码用于发送普通的 ASCII 字符，它将一串 7-bit 的字符（最高位为 0）编码成 8-bit 的数据，每 8 个字符可“压缩”成 7 个；8-bit 编码通常用于发送数据消息，如图片和铃声等；而 UCS2 编码用于发送 Unicode 字符。PDU 串的用户信息（TP-UD）段最大容量是 140 字节，所以在这 3 种编码方式下，可以发送的短消息的最大字符数分别是 160、140 和 70。这里，将一个英文字母、一个汉字和一个数据字节都视为一个字符。需要注意的是，PDU 串的用户信息长度（TP- UDL），在各种编码方式下意义有所不同：7-bit 编码时：指原始短消息的字符个数，而不是编码后的字节数；8-bit 编码时，就是字节数；UCS2 编码时，也是字节数，等于原始短消息的字符数的两倍。

7.5　本 章 小 结

本章介绍了如何实现短信的接收和发送，通过调用 Android 系统本身的短信管理类对短信进行管理，同时了解了短信发送过程的一些细节和注意事项。短信功能作为手机的一个基本功能，一般手机都会自带短信管理软件，但是了解短信的收发机制，可以帮助我们在程序中嵌入短信收发的功能，或者增强系统自身的短信功能。

第 8 章　通　讯　录

每个人的手机中必不可少的一个软件是什么呢？是的，就是通讯录。无法想象没有通讯录的手机还怎么使用，通讯录可以和很多功能整合到一起，如短信、电话等，形成一个集中化管理的软件。本章要介绍的通讯录是一个最简单的通讯录，但却是通讯录功能的核心。

8.1　界 面 设 计

本章要实现的通讯录功能主要有：联系人的添加、删除、编辑、查看，向选中的联系人打电话、发信息等。程序首先呈现在我们眼前的是主页面，在主页面中以一个 ListView 显示当前添加的所有的通讯录，如图 8.1 所示。

添加联系人和编辑联系人的界面，如图 8.2 所示。



图 8.1　通讯录主界面



图 8.2　添加联系人

单击主页面中任意一个名字，即可查看对应的通讯录的详细信息，如图 8.3 所示。



图 8.3　查看联系人信息

以上就是本程序需要呈现的 3 个不同的视图，第一个视图不需要布局文件，直接在程序中实现界面。下面我们来介绍如何设计第二个视图，也就是添加联系人的布局文件。

8.1.1　布局的设置

整个布局的主体采用 TableRow 来呈现，采用 TableRow 的好处是对齐很方便，适合于表格布局。每一行都用一个 TextView 的标签加一个 EditText 用于提供用户填写信息，最后一行放置两个按钮，一个“确定”按钮，一个“取消”按钮。

```
001 <?xml version="1.0" encoding="utf-8"?>
002 <LinearLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
003    android:orientation="vertical"
004    android:layout width="fill parent"
005    android:layout height="fill parent"
006    android:background="@drawable/bg">
007    <TableRow
008        android:id="@+id/TableRow01"
009        android:layout width="fill parent"
010        android:layout_height="wrap_content">
011        <!-- 姓名标签 -->
012        <TextView
013            android:id="@+id/TextView01"
014            android:layout width="wrap content"
015            android:layout height="wrap content"
016            android:text="@string/name"
017            android:textSize="16px" />
```

```
018         <!-- 姓名 -->
019         <EditText
020             android:id="@+id/EditText01"
021             android:layout_height="wrap_content"
022             android:layout_width="fill_parent" />
023     </TableRow>
024     <TableRow
025         android:id="@+id/TableRow02"
026         android:layout_width="fill_parent"
027         android:layout_height="wrap_content">
028         <!-- 手机标签 -->
029         <TextView
030             android:id="@+id/TextView02"
031             android:layout_width="wrap_content"
032             android:layout_height="wrap_content"
033             android:textSize="16px"
034             android:text="@string/mobile" />
035         <!-- 手机 -->
036         <EditText
037             android:id="@+id/EditText02"
038             android:layout_height="wrap_content"
039             android:layout_width="fill_parent"
040             android:inputType="phone" />
041     </TableRow>
042     <TableRow
043         android:id="@+id/TableRow03"
044         android:layout_width="fill_parent"
045         android:layout_height="wrap_content">
046         <!-- 座机标签 -->
047         <TextView
048             android:id="@+id/TextView03"
049             android:layout_width="wrap_content"
050             android:layout_height="wrap_content"
051             android:textSize="16px"
052             android:text="@string/home" />
053         <!-- 座机 -->
054         <EditText
055             android:id="@+id/EditText03"
056             android:layout_height="wrap_content"
057             android:layout_width="fill_parent"
058             android:inputType="phone" />
059     </TableRow>
060     <TableRow
061         android:id="@+id/TableRow04"
062         android:layout_width="fill_parent"
063         android:layout_height="wrap_content">
064         <!-- 住址标签 -->
065         <TextView
066             android:id="@+id/TextView04"
067             android:layout_width="wrap_content"
068             android:layout_height="wrap_content"
069             android:textSize="16px"
070             android:text="@string/address" />
071         <!-- 住址 -->
072         <EditText
073             android:id="@+id/EditText04"
074             android:layout_height="wrap_content"
075             android:layout_width="fill_parent" />
076     </TableRow>
077     <TableRow
```

```
078         android:id="@+id/TableRow05"
079         android:layout width="fill parent"
080         android:layout height="wrap content">
081         <!-- 电子邮箱标签 -->
082         <TextView
083             android:id="@+id/TextView05"
084             android:layout_width="wrap_content"
085             android:layout_height="wrap_content"
086             android:text="@string/email"
087             android:textSize="16px" />
088         <!-- 电子邮箱 -->
089         <EditText
090             android:id="@+id/EditText05"
091             android:layout height="wrap content"
092             android:layout width="fill parent"
093             android:inputType="textEmailAddress" />
094     </TableRow>
095     <TableRow
096         android:id="@+id/TableRow07"
097         android:layout_width="fill_parent"
098         android:layout_height="wrap_content"
099         android:gravity="center_horizontal" >
100         <!—“保存”按钮 -->
101         <Button
102             android:id="@+id/Button01"
103             android:layout height="wrap content"
104             android:text="@string/ok"
105             android:textSize="16px"
106             android:layout width="wrap content" />
107         <!—“取消”按钮 -->
108         <Button
109             android:id="@+id/Button02"
110             android:layout_height="wrap_content"
111             android:text="@string/cancel"
112             android:textSize="16px"
113             android:layout width="wrap content" />
114     </TableRow>
115 </LinearLayout>
```

8.1.2　添加“查看联系人”页面

查看联系人的页面与添加页面很类似，基本就是把显示的 EditText 控件换成了 TextView 控件，并且在每一行之间用一个背景颜色为白色的 View 做成的分隔线隔开。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <TableLayout xmlns:android="http://schemas.android.com/
     apk/res/android"
03      android:layout width="fill parent"
04      android:layout height="fill parent"
05      android:stretchColumns="1"
06      android:background="@drawable/bg">
07      <TableRow>
08          <TextView
09              android:layout_column="1"
10              android:text="查看联系人"
11              android:textColor="#ff44f3ff"
12              android:padding="15dip" />
13      </TableRow>
```

```
14      <!-- 间隔线 -->
15      <View
16          android:layout height="3dip"
17          android:background="#ffffffff" />
18      <TableRow>
19          <!-- 姓名标签 -->
20          <TextView
21              android:layout column="1"
22              android:text="姓名: "
23              android:textColor="#ffffaaaf"
24              android:padding="10dip" />
25          <!-- 姓名 -->
26          <TextView
27              android:id="@+id/TextView Name"
28              android:textColor="#ffffaaaf"
29              android:padding="10dip" />
30      </TableRow>
31      <View
32          android:layout height="1dip"
33          android:background="#ffffffff" />
34      <TableRow>
35          <!-- 手机标签 -->
36          <TextView
37              android:layout_column="1"
38              android:text="手机: "
39              android:textColor="#ffffaaaf"
40              android:padding="10dip" />
41          <!-- 手机 -->
42          <TextView
43              android:id="@+id/TextView_Mobile"
44              android:textColor="#ffffaaaf"
45              android:padding="10dip" />
46      </TableRow>
47      <View
48          android:layout height="1dip"
49          android:background="#ffffffff" />
50      <TableRow>
51          <!-- 座机标签 -->
52          <TextView
53              android:layout column="1"
54              android:text="座机: "
55              android:textColor="#ffffaaaf"
56              android:padding="10dip" />
57          <!-- 座机 -->
58          <TextView
59              android:id="@+id/TextView Home"
60              android:textColor="#ffffaaaf"
61              android:padding="10dip" />
62      </TableRow>
63      <View
64          android:layout_height="1dip"
65          android:background="#ffffffff" />
66      <TableRow>
67          <!-- 住址标签 -->
68          <TextView
69              android:layout column="1"
70              android:text="地址: "
71              android:textColor="#ffffaaaf"
72              android:padding="10dip" />
```

```
73          <!-- 住址 -->
74          <TextView
75              android:id="@+id/TextView_Address"
76              android:textColor="#ffffaaaf"
77              android:padding="10dip" />
78      </TableRow>
79      <View
80          android:layout_height="1dip"
81          android:background="#ffffffff" />
82      <TableRow>
83          <!-- 电子邮箱标签 -->
84          <TextView
85              android:layout_column="1"
86              android:text="邮箱: "
87              android:textColor="#ffffaaaf"
88              android:padding="10dip" />
89          <!-- 电子邮箱 -->
90          <TextView
91              android:id="@+id/TextView_Email"
92              android:textColor="#ffffaaaf"
93              android:padding="10dip" />
94      </TableRow>
95      <View
96          android:layout_height="1dip"
97          android:background="#ffffffff" />
```

8.2　功 能 实 现

界面设计完后，下面开始实现通讯录的功能，具体有哪些功能请看下面的讲解。

8.2.1　创建数据库

因为通讯录的信息需要保存在数据库中，所以我们首先要创建一个数据库，新建DBHelper.java，代码如下所示。创建一个数据库，数据库文件名称为mycontacts.db，数据表名称为contacts，数据表包含的字段有“姓名”、“手机”、“座机”、“地址”和“电子邮箱”。

```
01  package com.guo.MyContacts;          //声明包语句
02~06 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
07  public class DBHelper extends SQLiteOpenHelper
08  {
09      //数据库名
10      public static final String DATABASE_NAME = "mycontacts.db";
11      //版本
12      public static final int DATABASE_VERSION = 2;
13      //表名
14      public static final String CONTACTS_TABLE = "contacts";
15      //创建表
16      private static final String DATABASE_CREATE =
17          "CREATE TABLE " + CONTACTS_TABLE +" ("
```

```
18          + ContactColumn. ID+" integer primary key autoincrement,"
19          + ContactColumn.NAME+" text,"
20          + ContactColumn.MOBILENUM+" text,"
21          + ContactColumn.HOMENUM+" text,"
22          + ContactColumn.ADDRESS+" text,"
23          + ContactColumn.EMAIL+" text )";
24      public DBHelper(Context context)
25      {
26          super(context, DATABASE_NAME, null, DATABASE_VERSION);
27      }
28      //创建数据库
29      public void onCreate(SQLiteDatabase db)
30      {
31          db.execSQL(DATABASE_CREATE);
32      }
33      //升级
34      public void onUpgrade(SQLiteDatabase db, int oldVersion, int
        newVersion)
35      {
36          db.execSQL("DROP TABLE IF EXISTS " + CONTACTS_TABLE);
37          onCreate(db);
38      }
```

8.2.2　创建 ContactColumn 类

我们新建一个类 ContactColumn，专门用于存放数据表的列名和索引值，代码如下所示：

```
01  package com.guo.MyContacts;
02
03  import android.provider.BaseColumns;
04
05  //定义数据
06  public class ContactColumn implements BaseColumns
07  {
08      public ContactColumn()
09      {
10      }
11      //列名
12      public static final String NAME = "name";                //姓名
13      public static final String MOBILENUM = "mobileNumber";  //移动电话
14      public static final String HOMENUM = "homeNumber";      //座机
15      public static final String ADDRESS = "address";         //地址
16      public static final String EMAIL = "email";             //邮箱
17      //列索引值
18      public static final int  ID_COLUMN = 0;
19      public static final int NAME_COLUMN = 1;
20      public static final int MOBILENUM_COLUMN = 2;
21      public static final int HOMENUM_COLUMN = 3;
22      public static final int ADDRESS_COLUMN = 4;
23      public static final int EMAIL_COLUMN = 5;
24
25      //查询结果
26      public static final String[] PROJECTION ={
27           ID,
28          NAME,
29          MOBILENUM,
```

```
30           HOMENUM,
31           ADDRESS,
32           EMAIL
33       };
```

8.2.3　为数据库提供操作类

创建了数据库之后，下一步自然需要为数据库提供操作类。新建 ContactsProvider.java 代码如下所示，在 ContactsProvider.java 中新建类 ContactsProvider 继承于 ContentProvider。代码 024～026 行设置了 ContentProvider 用于操作的 uri 地址，这些 uri 地址可以直接定位到指定的数据。代码 029～040 行定义了 uri 的匹配类型，有 CONTACTS 和 CONTACT_ID 两种，其中“#”号表示匹配任意字符，因此若匹配带 CONTATS 则表示匹配到所有联系人信息，而 CONTACT_ID 则表示对指定 id 的联系人进行操作。

在 onCreate 中初始化一些变量，接下去主要要实现 insert()、update()、delete()、query()这几个函数。在 delete()函数中，首先通过 uriMacher.match(uri)对传递过来的 uri 地址进行匹配，分情况进行处理，最后要通知更改并返回删除的个数。Insert 函数也类似，先判断 uri 地址是否合法，这时候 uri 地址必须匹配 CONTACTS，接着实例化一个 ContentValues，并将参数的值传递进去，若没有相应的值，则相应的值设置为“”。如果成功插入数据，则返回回该数据的 uri 地址。在 query 函数中新建一个 SQLiteQueryBuilder 变量用于执行查询，若传递的是 CONTACT_ID，则将相应的判断条件加入 where 语句，如代码 161 行所示，最后通过 qb.query 执行查询操作，返回游标。update 操作与 delete()函数类似，只不过操作的函数从 delete 变成了 update。

最后，代码 080～093 行实现了 getType()函数，主要用来根据 uri 地址返回相应的 MIME 类型，以调用对应的 Activity。

```
001 package com.guo.MyContacts;                    //声明包语句
002~014 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
015 public class ContactsProvider extends ContentProvider
016 {
017     //标签
018     private static final String TAG= "ContactsProvider";
019     //数据库帮助类
020     private DBHelper dbHelper;
021     //数据库
022     private SQLiteDatabase contactsDB;
023     //数据库操作 uri 地址
024     public static final String AUTHORITY = "com.guo.provider.
        ContactsProvider";
025     public static final String CONTACTS_TABLE = "contacts";
026     public static final Uri CONTENT_URI = Uri.parse("content://" +
        AUTHORITY + "/"+CONTACTS_TABLE);
027
028     //下面是自定义的类型
029     public static final int CONTACTS = 1;
030     public static final int CONTACT_ID = 2;
031     private static final UriMatcher uriMatcher;
032     static
```

```
033     {
034         //没有匹配的信息
035         uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
036         //全部联系人信息
037         uriMatcher.addURI(AUTHORITY,"contacts",CONTACTS);
038         //单独一个联系人信息
039         uriMatcher.addURI(AUTHORITY,"contacts/#",CONTACT_ID);
040     }
041     //取得数据库
042     @Override
043     public boolean onCreate()
044     {
045         dbHelper = new DBHelper(getContext());
046         //执行创建数据库
047         contactsDB = dbHelper.getWritableDatabase();
048         return (contactsDB == null) ? false : true;
049     }
050 
051     //删除指定的数据列
052     @Override
053     public int delete(Uri uri, String where, String[] selectionArgs)
054     {
055         int count;
056         switch (uriMatcher.match(uri))
057         {
058             //删除满足 where 条件的行
059             case CONTACTS:
060                 count = contactsDB.delete(CONTACTS_TABLE, where,
                    selectionArgs);
061                 break;
062             case CONTACT_ID:
063                 //取得联系人的 id 信息
064                 String contactID = uri.getPathSegments().get(1);
065                 //删除满足 where 条件，并且 id 值为 contactID 的记录
066                 count = contactsDB.delete(CONTACTS_TABLE,
067                                           ContactColumn._ID
068                                           + "=" + contactID
069                                           + (!TextUtils.isEmpty(where) ? "
                                          AND (" + where + ")" : ""),
070                                           selectionArgs);
071                 break;
072             default:
073                 throw new IllegalArgumentException("Unsupported URI: " +
                    uri);
074         }
075         getContext().getContentResolver().notifyChange(uri, null);
076         return count;
077     }
078 
079     //URI 类型转换
080     public String getType(Uri uri)
081     {
082         switch (uriMatcher.match(uri))
083         {
084             //所有联系人
085             case CONTACTS:
086                 return "vnd.android.cursor.dir/vnd.guo.android.
                    mycontacts";
087             //指定联系人
```

```
088             case CONTACT_ID:
089                 return "vnd.android.cursor.item/vnd.guo.
                    android.mycontacts";
090             default:
091                 throw new IllegalArgumentException("Unsupported URI: " +
                    uri);
092         }
093     }
094
095     //插入数据
096     public Uri insert(Uri uri, ContentValues initialValues)
097     {
098         //判断 uri 地址是否合法
099         if (uriMatcher.match(uri) != CONTACTS)
100         {
101             throw new IllegalArgumentException("Unknown URI " + uri);
102         }
103         ContentValues values;
104         if (initialValues != null)
105         {
106             values = new ContentValues(initialValues);
107             Log.e(TAG + "insert", "initialValues is not null");
108         }
109         else
110         {
111             values = new ContentValues();
112         }
113         //如果对应的名称没有值，则设置默认值为""
114         if (values.containsKey(ContactColumn.NAME) == false)
115         {
116             values.put(ContactColumn.NAME, "");
117         }
118         if (values.containsKey(ContactColumn.MOBILENUM) == false)
119         {
120             values.put(ContactColumn.MOBILENUM, "");
121         }
122         if (values.containsKey(ContactColumn.HOMENUM) == false)
123         {
124             values.put(ContactColumn.HOMENUM, "");
125         }
126         if (values.containsKey(ContactColumn.ADDRESS) == false)
127         {
128             values.put(ContactColumn.ADDRESS, "");
129         }
130         if (values.containsKey(ContactColumn.EMAIL) == false)
131         {
132             values.put(ContactColumn.EMAIL, "");
133         }
134         Log.e(TAG + "insert", values.toString());
135         //插入数据
136         long rowId = contactsDB.insert(CONTACTS_TABLE, null, values);
137         if (rowId > 0)
138         {
139             //将 id 值加入 uri 地址中
140             Uri noteUri = ContentUris.withAppendedId(CONTENT_URI,
                rowId);
141             //通知改变
142             getContext().getContentResolver().notifyChange(noteUri,
                null);
143             Log.e(TAG + "insert", noteUri.toString());
```

```
144             return noteUri;
145         }
146         throw new SQLException("Failed to insert row into " + uri);
147     }
148
149     //查询数据
150     public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder)
151     {
152         Log.e(TAG + ":query", " in Query");
153         SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
154         //设置要查询的数据表
155         qb.setTables(CONTACTS_TABLE);
156
157         switch (uriMatcher.match(uri))
158         {
159             //构建 where 语句，定位到指定 id 值的列
160             case CONTACT_ID:
161                 qb.appendWhere(ContactColumn._ID + "=" + uri.
                    getPathSegments().get(1));
162                 break;
163             default:
164                 break;
165         }
166         //查询
167         Cursor c = qb.query(contactsDB, projection, selection,
            selectionArgs, null, null, sortOrder);
168         //设置通知改变的 uri
169         c.setNotificationUri(getContext().getContentResolver(), uri);
170         return c;
171     }
172
173     //更新数据库
174     public int update(Uri uri, ContentValues values, String where,
        String[] selectionArgs)
175     {
176         int count;
177         Log.e(TAG + "update", values.toString());
178         Log.e(TAG + "update", uri.toString());
179         Log.e(TAG + "update :match", "" + uriMatcher.match(uri));
180         switch (uriMatcher.match(uri))
181         {
182             //根据 where 条件批量进行更新
183             case CONTACTS:
184                 Log.e(TAG + "update", CONTACTS + "");
185                 count = contactsDB.update(CONTACTS_TABLE, values, where,
                    selectionArgs);
186                 break;
187             //更新指定行
188             case CONTACT_ID:
189                 String contactID = uri.getPathSegments().get(1);
190                 Log.e(TAG + "update", contactID + "");
191                 count = contactsDB.update(CONTACTS_TABLE, values,
                    ContactColumn._ID + "=" + contactID
192                         + (!TextUtils.isEmpty(where) ? " AND (" + where +
                        ")" : ""), selectionArgs);
193                 break;
194             default:
195                 throw new IllegalArgumentException("Unsupported URI: " +
                    uri);
```

```
196         }
197         //通知更改
198         getContext().getContentResolver().notifyChange(uri, null);
199         return count;
200     }
```

8.2.4　ListView 界面的实现

分析完数据存储方式，接下去我们按照程序运行的顺序来逐个界面进行分析。首先是主界面，新建 MyContacts.java，继承于 ListActivity，因为我们要实现的是一个 ListView 的界面。代码 03～05 行定义了菜单的 id 值，代码 10 行的作用是设置当前按键的模式，以及当用户按键的时候会触发当前菜单中设置的快捷键功能。接着代码 17 行设置列表长按监听器，21～30 行查询联系人数据库中所有的数据，并将数据绑定到 adapter 中，最后为当前界面设置适配器 adapter。

```
01 public class MyContacts extends ListActivity
02 {
03     private static final int AddContact_ID = Menu.FIRST;
04     private static final int DELEContact_ID = Menu.FIRST+2;
05     private static final int EXITContact_ID = Menu.FIRST+3;
06 
07     public void onCreate(Bundle savedInstanceState)
08     {
09         super.onCreate(savedInstanceState);
10         setDefaultKeyMode(DEFAULT_KEYS_SHORTCUT);
11         //为 intent 绑定数据
12         Intent intent = getIntent();
13         if (intent.getData() == null) {
14             intent.setData(ContactsProvider.CONTENT_URI);
15         }
16         //设置菜单项长按监听器
17         getListView().setOnCreateContextMenuListener(this);
18         //设置背景图片
19         getListView().setBackgroundResource(R.drawable.bg);
20         //查询，获得所有联系人的数据
21         Cursor cursor = managedQuery(getIntent().getData(),
           ContactColumn.PROJECTION, null, null,null);
22 
23         //注册每个列表表示形式 ： 姓名 + 移动电话
24         SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
25             android.R.layout.simple_list_item_2,
26             cursor,
27             new String[] {ContactColumn.NAME, ContactColumn.MOBILENUM },
28             new int[] { android.R.id.text1, android.R.id.text2 });
29 
30         setListAdapter(adapter);
31     }
```

8.2.5　创建菜单

作为一个 List 菜单，我们通常需要做的事情就是为按键绑定监听器，分为长按和单击，当然，还包括任何界面都可以使用的创建菜单。

如下所示，代码 02～33 行为 menu 按键设置监听器，并为菜单的按键绑定监听器。当单击 menu 按键的时候，程序将创建两个菜单选项，一个是添加联系人，一个是退出，它们的快捷键分别是“a”和“d”。

代码 36～41 行设置列表选项的单击功能：查看联系人信息。44～93 行为长按选项监听器，当用户长按一个选项时，将弹出删除当前联系人的菜单，用户可以通过这种方式删除指定的联系人。onCreateContextMenu 为长按菜单项触发的函数，而 onContextItemSelected 则为菜单项按键监听器。

```
01  //添加菜单选项
02    public boolean onCreateOptionsMenu(Menu menu)
03    {
04        super.onCreateOptionsMenu(menu);
05        //添加联系人
06        menu.add(0, AddContact_ID, 0, R.string.add_user)
07          .setShortcut('3', 'a')
08          .setIcon(R.drawable.add);
09
10        //退出程序
11        menu.add(0, EXITContact_ID, 0, R.string.exit)
12          .setShortcut('4', 'd')
13          .setIcon(R.drawable.exit);
14        return true;
15
16    }
17
18    //处理菜单操作
19    public boolean onOptionsItemSelected(MenuItem item)
20    {
21        switch (item.getItemId())
22        {
23        case AddContact_ID:
24            //添加联系人
25            startActivity(new Intent(Intent.ACTION_INSERT, getIntent().
            getData()));
26            return true;
27        case EXITContact_ID:
28          //退出程序
29          this.finish();
30            return true;
31        }
32        return super.onOptionsItemSelected(item);
33    }
34    //动态菜单处理
35    //单击的默认操作也可以在这里处理
36    protected void onListItemClick(ListView l, View v, int position, long id)
37    {
38        Uri uri = ContentUris.withAppendedId(getIntent().getData(), id);
39      //查看联系人
40      startActivity(new Intent(Intent.ACTION_VIEW, uri));
41    }
42
43    //长按触发的菜单
44  public void onCreateContextMenu(ContextMenu menu, View view,
    ContextMenuInfo menuInfo)
45  {
46     AdapterView.AdapterContextMenuInfo info;
```

```
47      try
48      {
49          info = (AdapterView.AdapterContextMenuInfo) menuInfo;
50      }
51      catch (ClassCastException e)
52      {
53          return;
54      }
55      //得到长按的数据项
56      Cursor cursor = (Cursor) getListAdapter().getItem(info.position);
57      if (cursor == null)
58      {
59          return;
60      }
61
62      menu.setHeaderTitle(cursor.getString(1));
63      //添加删除菜单
64      menu.add(0, DELEContact ID, 0, R.string.delete user);
65  }
66    //长按列表触发的函数
67    @Override
68    public boolean onContextItemSelected(MenuItem item)
69  {
70      AdapterView.AdapterContextMenuInfo info;
71      try
72      {
73          //获得选中项的信息
74          info = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
75      }
76      catch (ClassCastException e)
77      {
78          return false;
79      }
80
81      switch (item.getItemId())
82      {
83          //删除操作
84          case DELEContact ID:
85          {
86              //删除一条记录
87              Uri noteUri = ContentUris.withAppendedId(getIntent().
                getData(), info.id);
88              getContentResolver().delete(noteUri, null, null);
89              return true;
90          }
91      }
92      return false;
93  }
```

8.2.6　实现界面的查看

跟主页面息息相关的两个界面，一个是查看界面，一个是编辑、添加界面。新建 ContactView 继承于 Activity，用于实现查看功能，代码如下所示。在 onCreate 中根据传递过来的 uri 地址获取联系人的信息，并将信息逐个显示到对应的界面位置中。然后，一如既往地，设置 menu 菜单，如代码 066～082 行所示。可以看到，这里我们设置了五个菜单项，第一个为删除当前联系人，将调用 deleteContact()函数删除当前的联系人，接着返回主

页面；第二个为返回列表，将关闭当前的 activity，返回到主页面；第三个为编辑联系人，也就是下面我们要介绍的页面；第四个、第五个分别为呼叫联系人和发信息，将调用系统程序进行拨号和发信息，这里要注意 Intent 的特殊写法。

```
001 package com.guo.MyContacts;                    //声明包语句
002~011 行为引入相关类，这里不再列举，请阅读光盘内容。
//
……
012 public class ContactView extends Activity
013 {
014     //姓名
015     private TextView mTextViewName;
016     //手机
017     private TextView mTextViewMobile;
018     //座机
019     private TextView mTextViewHome;
020     //住址
021     private TextView mTextViewAddress;
022     //电子邮箱
023     private TextView mTextViewEmail;
024
025     private Cursor mCursor;
026     private Uri mUri;
027     //设置菜单的序号
028     private static final int REVERT_ID = Menu.FIRST;
029     private static final int DELETE_ID = Menu.FIRST + 1;
030     private static final int EDITOR_ID = Menu.FIRST + 2;
031     private static final int CALL_ID = Menu.FIRST + 3;
032     private static final int SENDSMS_ID = Menu.FIRST + 4;
033
034     public void onCreate(Bundle savedInstanceState)
035     {
036         super.onCreate(savedInstanceState);
037         mUri = getIntent().getData();
038         this.setContentView(R.layout.viewuser);
039         //初始化界面元素
040         mTextViewName = (TextView) findViewById(R.id.TextView_Name);
041         mTextViewMobile = (TextView) findViewById(R.id.TextView_
            Mobile);
042         mTextViewHome = (TextView) findViewById(R.id.TextView_Home);
043         mTextViewAddress = (TextView) findViewById(R.id.TextView_
            Address);
044         mTextViewEmail = (TextView) findViewById(R.id.TextView_Email);
045
046         //获得并保存原始联系人信息
047         mCursor = managedQuery(mUri, ContactColumn.PROJECTION, null,
            null, null);
048         mCursor.moveToFirst();
049         if (mCursor != null)
050         {
051             //读取并显示联系人信息
052             mCursor.moveToFirst();
053
054             mTextViewName.setText(mCursor.getString(ContactColumn.
            NAME_COLUMN));
055             mTextViewMobile.setText(mCursor.getString(ContactColumn.
             MOBILENUM_COLUMN));
056             mTextViewHome.setText(mCursor.getString(ContactColumn.
```

```
                HOMENUM COLUMN));
057             mTextViewAddress.setText(mCursor.getString(ContactColumn.
                ADDRESS COLUMN));
058             mTextViewEmail.setText(mCursor.getString(ContactColumn.
                EMAIL COLUMN));
059         }
060         else
061         {
062             setTitle("错误信息");
063         }
064     }
065     //添加菜单
066     public boolean onCreateOptionsMenu(Menu menu)
067     {
068         super.onCreateOptionsMenu(menu);
069         //返回
070         menu.add(0, REVERT_ID, 0, R.string.revert).setShortcut('0',
           'r').setIcon(R.drawable.listuser);
071         //删除联系人
072         menu.add(0, DELETE ID, 0, R.string.delete user).setShortcut('0',
            'd').setIcon(R.drawable.remove);
073         //编辑联系人
074         menu.add(0, EDITOR_ID, 0, R.string.editor_user).setShortcut('0',
            'd').setIcon(R.drawable.edituser);
075         //呼叫用户
076         menu.add(0, CALL ID, 0, R.string.call user).setShortcut('0',
            'd').setIcon(R.drawable.calluser)
077                 .setTitle(this.getResources().getString(R.string.call
                 user)+mTextViewName.getText());
078         //发送短信
079         menu.add(0, SENDSMS_ID, 0, R.string.sendsms_user).setShortcut
            ('0', 'd').setIcon(R.drawable.sendsms)
080         .setTitle(this.getResources().getString(R.string.sendsms_user)
             +mTextViewName.getText());
081         return true;
082     }
083
084     public boolean onOptionsItemSelected(MenuItem item)
085     {
086         switch (item.getItemId())
087         {
088             //删除
089             case DELETE_ID:
090                 deleteContact();
091                 finish();
092                 break;
093             //返回列表
094             case REVERT ID:
095                 setResult(RESULT CANCELED);
096                 finish();
097                 break;
098             case EDITOR ID:
099             //编辑联系人
100                 startActivity(new Intent(Intent.ACTION_EDIT, mUri));
101                 finish();
102                 break;
103             case CALL_ID:
104             //呼叫联系人
105                 Intent call = new Intent(Intent.ACTION_CALL,Uri.parse
```

```
                      ("tel:"+mTextViewMobile.getText()));
106                 startActivity(call);
107                 break;
108             case SENDSMS_ID:
109             //发短信给联系人
110                 Intent sms = new Intent(Intent.ACTION_SENDTO,Uri.parse
                    ("smsto:"+mTextViewMobile.getText()));
111                 startActivity(sms);
112                 break;
113         }
114         return super.onOptionsItemSelected(item);
115     }
116
117     //删除联系人信息
118     private void deleteContact()
119     {
120         if (mCursor != null)
121         {
122             mCursor.close();
123             mCursor = null;
124             getContentResolver().delete(mUri, null, null);
125             setResult(RESULT_CANCELED);
126         }
127     }
```

8.2.7　添加一个标志变量

首先，大家要清楚的一点是，这个界面是添加和编辑公用的界面，因此为了区分这两种情况，我们需要使用一个标志变量来进行区分。以下代码 012 行声明的 mState 变量即为标志变量，它可以取 STATE_EDIT 和 STATE_INSERT 这两个值。

这个界面的初始化函数 onCreate()是最关键的部分，因为在这里需要决定如何显示。代码 032～054 行根据 intent 传递的 action 值判断当前处于编辑还是新建状态，并将状态保存到 mState 中，以便于接下去的一系列操作。实际上，我们的添加操作分为两步，第一步是当我们进入这个界面的时候，我们已经新建了一个只包含 id 信息的数据，第二步是对这个数据进行更新操作。因此在代码 98～127 行都是从数据库取得相应联系人的信息，然后填充到指定的文本框。

代码 066～096 行分别设置了两个按钮的监听器，单击“保存”的时候需要判断是否输入了东西，如果没有输入，则将原来的记录删除，否则更新数据。单击“取消”按钮则直接删除当前的数据，并关闭当前页面。

```
001 public class ContactEditor extends Activity
002 {
003     //标志位常量，用于标志当前是新建状态还是编辑状态
004     private static final int STATE_EDIT = 0;
005     private static final int STATE_INSERT = 1;
006
007     private static final int REVERT_ID = Menu.FIRST;
008     private static final int DISCARD_ID = Menu.FIRST + 1;
009     private static final int DELETE_ID = Menu.FIRST + 2;
010
011     private Cursor mCursor;
012     private int mState;     //当前处于新建状态还是编辑状态的标志位变量
013     private Uri mUri;
```

```
014     //界面元素
015     private EditText nameText;
016     private EditText mobileText;
017     private EditText homeText;
018     private EditText addressText;
019     private EditText emailText;
020     //按键
021     private Button okButton;
022     private Button cancelButton;
023
024     public void onCreate(Bundle savedInstanceState)
025     {
026         super.onCreate(savedInstanceState);
027
028         final Intent intent = getIntent();
029         final String action = intent.getAction();
030         //根据 action 的不同进行不同的操作
031         //编辑联系人
032         if (Intent.ACTION_EDIT.equals(action))
033         {
034             mState = STATE_EDIT;
035             mUri = intent.getData();
036         }
037         else if (Intent.ACTION_INSERT.equals(action))
038         {
039             //添加新联系人
040             mState = STATE_INSERT;
041             mUri = getContentResolver().insert(intent.getData(), null);
042             if (mUri == null)
043             {
044                 finish();
045                 return;
046             }
047             setResult(RESULT_OK, (new Intent()).setAction(mUri.
                toString()));
048         }
049         //其他情况，退出
050         else
051         {
052             finish();
053             return;
054         }
055         setContentView(R.layout.editorcontacts);
056         //初始化界面文本框
057         nameText = (EditText) findViewById(R.id.EditText01);
058         mobileText = (EditText) findViewById(R.id.EditText02);
059         homeText = (EditText) findViewById(R.id.EditText03);
060         addressText = (EditText) findViewById(R.id.EditText04);
061         emailText = (EditText) findViewById(R.id.EditText05);
062         //初始化按钮
063         okButton = (Button)findViewById(R.id.Button01);
064         cancelButton = (Button)findViewById(R.id.Button02);
065         //设置“确定”按钮监听器
066         okButton.setOnClickListener(new OnClickListener()
067         {
068             public void onClick(View v)
069             {
070                 String text = nameText.getText().toString();
071                 if(text.length() == 0)
072                 {
```

```
073                     //如果没有输入东西，则将原来的记录删除
074                     setResult(RESULT CANCELED);
075                     deleteContact();
076                     finish();
077                 }
078                 else
079                 {
080                     //更新数据
081                     updateContact();
082                 }
083             }
084         });
085         //设置“取消”按钮监听器
086         cancelButton.setOnClickListener(new OnClickListener()
087         {
088             public void onClick(View v)
089             {
090                 //不添加记录，也不保存记录
091                 setResult(RESULT CANCELED);
092                 deleteContact();
093                 finish();
094
095             }
096         });
097         //获得并保存原始联系人信息
098         mCursor = managedQuery(mUri, ContactColumn.PROJECTION, null,
            null, null);
099         mCursor.moveToFirst();
100         if (mCursor != null)
101         {
102             //读取并显示联系人信息
103             mCursor.moveToFirst();
104             if (mState == STATE EDIT)
105             {
106                 setTitle(getText(R.string.editor_user));
107             }
108             else if (mState == STATE_INSERT)
109             {
110                 setTitle(getText(R.string.add_user));
111             }
112             String name = mCursor.getString(ContactColumn.NAME_COLUMN);
113             String moblie = mCursor.getString(ContactColumn.
                MOBILENUM COLUMN);
114             String home = mCursor.getString(ContactColumn.
                HOMENUM COLUMN);
115             String address = mCursor.getString(ContactColumn.
                ADDRESS COLUMN);
116             String email = mCursor.getString(ContactColumn.
                EMAIL COLUMN);
117             //显示信息
118             nameText.setText(name);
119             mobileText.setText(moblie);
120             homeText.setText(home);
121             addressText.setText(address);
122             emailText.setText(email);
123         }
124         else
125         {
126             setTitle("错误信息");
127         }
```

8.2.8　设置 menu 菜单

接着要设置 menu 菜单，以下代码 02～24 行分两种情况为界面添加菜单，代码 27～47 行则分别为菜单的按键设置功能。可以看到，当为编辑模式时，创建的菜单包含“返回”和“删除联系人”两个选项。当为新建模式时，只有一个“返回”选项。

接下去的代码主要实现了删除和更新的功能，主要是调用我们新建的 ContentProvier 的 delete()和 update()函数。

```
01 //菜单选项
02   public boolean onCreateOptionsMenu(Menu menu)
03   {
04       super.onCreateOptionsMenu(menu);
05       if (mState == STATE_EDIT)
06       {
07         //“返回”按钮
08           menu.add(0, REVERT_ID, 0, R.string.revert)
09                   .setShortcut('0', 'r')
10                   .setIcon(R.drawable.listuser);
11           //“删除联系人”按钮
12           menu.add(0, DELETE_ID, 0, R.string.delete_user)
13           .setShortcut('0', 'f')
14           .setIcon(R.drawable.remove);
15       }
16       else
17       {
18         //“返回”按钮
19           menu.add(0, DISCARD_ID, 0, R.string.revert)
20                   .setShortcut('0', 'd')
21                   .setIcon(R.drawable.listuser);
22       }
23       return true;
24   }
25   //菜单处理
26 @Override
27   public boolean onOptionsItemSelected(MenuItem item)
28 {
29       switch (item.getItemId())
30       {
31       //删除联系人
32       case DELETE_ID:
33         deleteContact();
34           finish();
35           break;
36        //删除刚创建的空联系人
37       case DISCARD_ID:
38         cancelContact();
39         finish();
40           break;
41       //直接返回
42       case REVERT_ID:
43         finish();
44           break;
45       }
46       return super.onOptionsItemSelected(item);
47   }
```

```
48
49  //删除联系人信息
50  private void deleteContact()
51  {
52      if (mCursor != null)
53      {
54           mCursor.close();
55           mCursor = null;
56           getContentResolver().delete(mUri, null, null);
57           nameText.setText("");
58        }
59  }
60  //丢弃信息
61  private void cancelContact()
62  {
63      if (mCursor != null)
64      {
65          deleteContact();
66        }
67        setResult(RESULT_CANCELED);
68        finish();
69  }
70  //更新变更的信息
71  private void updateContact()
72  {
73      if (mCursor != null)
74      {
75          mCursor.close();
76           mCursor = null;
77           ContentValues values = new ContentValues();
78          values.put(ContactColumn.NAME, nameText.getText().toString());
79          values.put(ContactColumn.MOBILENUM, mobileText.getText().
            toString());
80          values.put(ContactColumn.HOMENUM, homeText.getText().
            toString());
81          values.put(ContactColumn.ADDRESS, addressText.getText().
            toString());
82          values.put(ContactColumn.EMAIL, emailText.getText().
            toString());
83          //更新数据
84           getContentResolver().update(mUri, values, null, null);
85        }
86        setResult(RESULT_CANCELED);
87        finish();
88  }
```

最后值得一提的是我们的 AndroidManifest.xml 的写法，代码如下所示。其中要注意的是，声明权限的部分以及查看、编辑、新建界面的写法，里面使用了 data 标签，这个我们之前比较少接触到，后面将会详细跟大家讲一下这个标签的使用。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03        package="com.guo.MyContacts"
04        android:versionCode="1"
05        android:versionName="1.0">
06        <!-- 声明打电话、收发短信的权限 -->
07      <uses-permission android:name="android.permission.CALL_PHONE" />
08      <uses-permission android:name="android.permission.SEND_SMS" />
09      <uses-permission android:name="android.permission.RECEIVE_SMS" />
```

```
10     <uses-sdk android:minSdkVersion="10" />
11     <application android:icon="@drawable/icon" android:label="@string/
      app_name">
12       <!-- 声明 contentProvider -->
13     <provider android:name="com.guo.MyContacts.ContactsProvider"
14             android:authorities="com.guo.provider.ContactsProvider"/>
15     <!-- 主界面 -->
16         <activity android:name=".MyContacts"
17                 android:label="@string/app_name">
18             <intent-filter>
19                 <action android:name="android.intent.action.MAIN" />
20                 <category android:name="android.intent.category.
                   LAUNCHER" />
21             </intent-filter>
22         </activity>
23         <!-- 编辑、新建界面 -->
24         <activity android:name=".ContactEditor"
25                 android:label="@string/editor_user">
26             <intent-filter>
27                 <action android:name="android.intent.action.EDIT" />
28                 <category android:name="android.intent.category.
                    DEFAULT" />
29                 <data android:mimeType="vnd.android.cursor.item/vnd.guo.
                    android.mycontacts" />
30             </intent-filter>
31             <intent-filter>
32                 <action android:name="android.intent.action.INSERT" />
33                 <category android:name="android.intent.category.
                    DEFAULT" />
34                 <data android:mimeType="vnd.android.cursor.dir/vnd.guo.
                    android.mycontacts" />
35             </intent-filter>
36         </activity>
37         <!-- 查看界面 -->
38         <activity android:name="com.guo.MyContacts.ContactView"
39                 android:label="@string/view_user">
40             <intent-filter>
41                 <action android:name="android.intent.action.VIEW" />
42                 <category android:name="android.intent.category.
                    DEFAULT" />
43                 <data android:mimeType="vnd.android.cursor.item/vnd.guo.
                   android.mycontacts" />
44             </intent-filter>
45             <intent-filter>
46                 <category android:name="android.intent.category.
                    DEFAULT" />
47                 <data android:mimeType="vnd.android.cursor.dir/vnd.guo.
                    android.mycontacts" />
48             </intent-filter>
49         </activity>
50     </application>
```

8.3　知识拓展

文章的最后我们谈到了 intent-filter 中的 data 标签，那么 data 标签有哪些属性呢？data 标签包含的属性有 scheme、host、port、path、pathPrefix 和 pathPattern，这些属性的作用都

是用来匹配 uri 的，书写形式为 scheme://host:port/pathorpathPrefixorpathPattern。其中比较关键的一个属性是最后的 pathorpathPrefixorpathPattern， path 用来匹配完整的路径，如：http://example.com/blog/abc.html，这里将 path 设置为 /blog/abc.html 才能够进行匹配；pathPrefix 用来匹配路径的开头部分，拿上面的 Uri 来说，这里将 pathPrefix 设置为 /blog 就能进行匹配了；pathPattern 用表达式来匹配整个路径。这里需要说一下匹配符号与转义。匹配符号：“*” 用来匹配 0 次或更多，如“a*” 可以匹配“a”、“aa”、“aaa”……。“.” 用来匹配任意字符，如“.” 可以匹配“a”、“b”，“c”……。因此“.*” 就是用来匹配任意字符 0 次或更多，如“.*html” 可以匹配 “abchtml”、“chtml”，“html”，“sdf.html”……；转义：因为当读取 Xml 的时候，“\”是被当作转义字符的（当它被用作 pathPattern 转义之前），因此这里需要两次转义，读取 Xml 是一次，在 pathPattern 中使用又是一次。如：“*” 这个字符就应该写成 “*”，“\” 这个字符就应该写成 “\\\\”。

我们先来看一个简单的应用。

如果我们想要匹配以.MP3 结尾的路径，使得我们的程序在打开网上的 mp3 文件时，能够选择我们的程序进行下载。我们可以将 scheme 设置为 http，pathPattern 设置为“.*\\.MP3”，整个 intent-filter 的写法如下所示。如果你想处理某个站点的 MP3，那么在 data 标签里面增加 android:host=“yoursite.com”。

```
<intent-filter>
<action android:name="android.intent.action.VIEW"></action>
<category android:name="android.intent.category.DEFAULT"></category>
<data android:scheme="http" android:pathPattern=".*\\.MP3"></data>
</intent-filter>
```

data 标签还有一个 mimeType 属性，该属性是用来匹配 Intent 的。例如当你使用 Intent.setType(“text/plain”)，那么系统将会匹配到所有注册 android:mimeType=“text/plain” 的 Activity。这里要注意的是 Intent.setType()、Intent.setData、Intent.setDataAndType() 这 3 个方法！

setType 调用后设置 mimeType，然后将 data 置为 null；setData 调用后设置 data，然后将 mimeType 置为 null；setDataAndType 调用后才会同时设置 data 与 mimeType。

另外需要注意的是，如果在 data 标签既设置了 mimeType 又设置了 scheme ，那么你的 Intent 需要同时设置匹配的 data 与 mimeType，即调用 setDataAndType ，系统才能匹配到这个 Activity（即便你 mimeType 设置为“*/*”也是如此）。当然，如果你没有设置 mimeType，那么调用 setData 进行匹配，如果你设置了任何的 mimeType 将不会匹配到该 Activity。

如果我们做的是一个 IM 应用，或是其他类似于微博之类的应用，如何让别人通过 Intent 调用，让我们的程序出现在弹出的选择框里呢？我们只用注册 android.intent.action.SEND 与 mimeType 为 text/plain 或 “*/*” 就可以了，整个 intent-filter 设置如下所示。这里设置 category 的原因是，创建的 Intent 的实例默认 category 就包含了 Intent.CATEGORY_DEFAULT，Google 这样做的原因是为了让这个 Intent 始终有一个 category。

```
1   <intent-filter>
2   <action android:name="android.intent.action.SEND"/>
3   <category android:name="android.intent.category.DEFAULT"/>
```

```
4   <data mimeType="*/*"/>
5   </intent-filter>
```

8.4　本章小结

本章介绍了如何开发一个简单的通讯录，从通讯录的添加、编辑、删除、查询这几个方面讲述了通讯录的开发过程。本章使用了 ContentProvider 进行数据操作，大家需要掌握 ContentProvider 的使用，包括如何实现回调函数，如何调用，如何在 AndroidManifest.xml 文件中注册。文章最后介绍了 intent-filter 中 data 标签的使用，大家需要熟练掌握 data 标签各个属性的含义。

第 9 章　任务管理器

装在手机上的软件越来越多，有一天发现手机变得卡的不行，于是乎，你想找出究竟是哪一个程序占用了你宝贵的资源。你想打开“任务管理器”看个究竟，却发现安卓完全没有 Windows 方便。那么，何不自己做一个呢？本章我们将要开发的就是一款类似于 Window 下任务管理器的软件。

9.1　功 能 分 析

任务管理器，顾名思义，我们要知道系统当前运行了哪些程序，这些程序都是做什么的，占用了多少资源。当然，必要的时候我们会想要结束掉某个进程。

如图 9.1 所示，是我们最终实现的效果图。以一个 ListView 显示当前运行的所有进程，每一行的左边显示程序的 icon，右边为 3 行文字，最上面为程序名，中间是包名，最下面是程序占用的内存大小。最下面设置两个按钮，分别是“刷新”和“全部结束”，刷新即刷新当前界面，全部结束将会结束所有用户程序（当然，本程序除外）。



图 9.1　任务管理器

此外，当用户单击列表元素时将会弹出对话框，如图 9.2 所示，可以对程序进程进一

步操作。左边是查看详情，可以查看应用程序一些更细节的信息，而强制结束则可以结束当前进程。



图 9.2　单击弹出对话框

9.2　界 面 设 计

9.1 节的内容介绍了任务管理器的常见功能，本节就讲解一下如何实现这些功能，具体如下。

9.2.1　编写主界面

主界面 main.xml 代码如下所示，最上方一个 ListView 用于显示所有进程的信息，最下面放置一个 RelativeLayout，里面放置了两个按钮，一个是“刷新”，另一个是“强制结束”。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:layout_height="wrap_content"
04      android:layout_width="fill_parent"
05      android:orientation="vertical"
06      android:scrollbars="vertical">
07      <!-- 用于显示所有进程 -->
08      <ListView android:id="@id/android:list"
09          android:layout_above="@+id/relativelayout_operation"
10          android:layout_width="fill_parent"
11          android:layout_height="wrap_content"/>
```

```
12      <!-- 位于界面下方的两个按钮 -->
13      <RelativeLayout
14          android:id="@+id/relativelayout_operation"
15          android:layout_width="fill_parent"
16          android:layout_height="wrap_content"
17          android:layout_alignParentBottom="true">
18          <!—“刷新”按钮，用于刷新界面 -->
19          <Button
20              android:id="@+id/btn_refresh_process"
21              android:text="刷新"
22              android:layout_height="wrap_content"
23              android:layout_width="wrap_content"
24              android:textSize="7pt"
25              android:paddingTop="1dip"
26              android:paddingBottom="1dip"
27              android:paddingLeft="20dip"
28              android:paddingRight="20dip"
29              android:layout_alignParentLeft="true"
30              android:layout_marginLeft="30dip" />
31          <!--“强制结束”按钮，用于结束所有用户进程 -->
32          <Button
33              android:id="@+id/btn_closeall_process"
34              android:text="@string/kill_all"
35              android:layout_width="wrap_content"
36              android:layout_height="wrap_content"
37              android:textSize="7pt"
38              android:paddingTop="1dip"
39              android:paddingBottom="1dip"
40              android:paddingLeft="20dip"
41              android:paddingRight="20dip"
42              android:layout_alignParentRight="true"
43              android:layout_marginRight="30dip" />
44      </RelativeLayout>
```

9.2.2　ListView 布局的设置

下面是 ListView 的元素布局，代码如下所示，最外面使用一个 RelativeLayout，以水平方向布局。左边是一个 ImageView 用于放置程序的 icon，右边是一个 LinearLayout 用于存放程序的描述信息。从上到下依次是程序的名称、包名、占用的内存。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout
03    xmlns:android="http://schemas.android.com/apk/res/android"
04    android:layout_width="fill_parent"
05    android:layout_height="wrap_content"
06    android:orientation="horizontal"
07    >
08      <!-- 用于放置程序的 icon -->
09      <ImageView
10          android:id="@+id/image_app"
11          android:layout_height="48dip"
12          android:layout_width="48dip"
13          android:layout_marginRight="10dip"
14          android:layout_alignParentLeft="true"
15      />
16      <!-- 显示描述进程信息的文字 -->
17      <LinearLayout
```

```
18          android:id="@+id/linearLayout1"
19          android:orientation="vertical"
20          android:layout_width="wrap_content"
21          android:layout_height="wrap_content"
22          android:layout_toRightOf="@id/image_app">
23          <!-- 程序的名称 -->
24          <TextView
25              android:text="123"
26              android:layout_height="wrap_content"
27              android:layout_width="wrap_content"
28              android:id="@+id/name_app"
29              android:textSize="8pt" />
30          <!-- 程序的包名 -->
31          <TextView
32              android:text="123"
33              android:layout_height="wrap_content"
34              android:layout_width="wrap_content"
35              android:id="@+id/package_app"
36              android:textSize="5pt" />
37          <!-- 程序占用的内存-->
38          <TextView
39              android:text="123"
40              android:id="@+id/cpumem_app"
41              android:textColor="#00AA00"
42              android:layout_height="wrap_content"
43              android:layout_width="wrap_content"
44              android:textSize="5pt" />
45      </LinearLayout>
46  </RelativeLayout>
```

9.2.3　显示程序的详细信息

最后一个布局文件用于显示程序的详细信息，代码如下所示。整个视图的根元素是一个 ScrollView，以标题和内容为分界逐个往下排。最上面用来显示程序的名称，同时右侧还有一个“强制结束”的按钮，用于结束当前进程。往下依次为版权信息、安装目录、数据目录、文件大小、操作权限、服务信息、活动信息。

```
001 <?xml version="1.0" encoding="utf-8"?>
002 <ScrollView
003     xmlns:android="http://schemas.android.com/apk/res/android"
004     android:layout_height="wrap_content"
005     android:layout_width="match_parent"
006     android:scrollbars="none">
007     <LinearLayout
008         android:id="@+id/linearlayout1"
009         android:layout_width="fill_parent"
010         android:layout_height="fill_parent"
011         android:orientation="vertical"
012         android:paddingLeft="3dip"
013         android:paddingRight="3dip"
014         android:paddingTop="1dip"
015         android:paddingBottom="1dip">
016         <!-- 显示进程名称和一个按钮 -->
017         <RelativeLayout android:layout_width="match_parent"
018             android:layout_height="27dip"
019             android:background="#555555"
020             android:id="@+id/relativeLayout1">
```

```
021             <!-- 进程名称==标题 -->
022             <TextView android:textSize="7pt"
023                android:layout alignParentLeft="true"
024                android:id="@+id/textView1"
025                android:layout_width="wrap_content"
026                android:layout_height="fill_parent"
027                android:gravity="center_vertical"
028                android:text="@string/detail_process_name" />
029             <!-- 强制结束==按钮 -->
030             <Button
031                android:id="@+id/btn kill process"
032                android:layout width="wrap content"
033                android:layout height="fill parent"
034                android:layout alignParentRight="true"
035                android:paddingBottom="0dip"
036                android:paddingLeft="10dip"
037                android:paddingRight="10dip"
038                android:paddingTop="0dip"
039                android:text="@string/kill process"
040                android:textColor="#cc0033"
041                android:textSize="5pt" />
042          </RelativeLayout>
043          <!--进程标题==内容  -->
044          <TextView
045             android:id="@+id/detail process name"
046             android:layout height="wrap content"
047             android:layout width="fill parent"
048             android:paddingTop="3dip"
049             android:paddingBottom="3dip" />
050          <!-- 版权信息==标题 -->
051          <TextView
052             android:text="@string/detail_process_copyright"
053             android:textSize="7pt"
054             android:layout_alignParentLeft="true"
055             android:layout width="fill parent"
056             android:layout height="25dip"
057             android:gravity="center vertical"
058             android:background="#555555"
059             android:paddingTop="3dip"
060             android:paddingBottom="3dip" />
061          <!-- 版权信息==内容 -->
062          <TextView
063             android:id="@+id/detail_process_copyright"
064             android:layout_height="wrap_content"
065             android:layout_width="fill_parent"
066             android:paddingTop="3dip"
067             android:paddingBottom="3dip" />
068          <!-- 安装目录==标题 -->
069          <TextView
070             android:text="@string/detail process install dir"
071             android:textSize="7pt"
072             android:layout alignParentLeft="true"
073             android:layout width="fill parent"
074             android:layout height="25dip"
075             android:gravity="center vertical"
076             android:background="#555555"
077             android:paddingTop="3dip"
078             android:paddingBottom="3dip" />
079          <!-- 安装目录==内容 -->
080          <TextView
```

```
081             android:id="@+id/detail_process_install_dir"
082             android:layout_height="wrap_content"
083             android:layout_width="fill_parent"
084             android:paddingTop="3dip"
085             android:paddingBottom="3dip" />
086         <!-- 数据目录==标题 -->
087         <TextView
088             android:text="@string/detail_process_data_dir"
089             android:textSize="7pt"
090             android:layout_alignParentLeft="true"
091             android:layout_width="fill_parent"
092             android:layout_height="25dip"
093             android:gravity="center_vertical"
094             android:background="#555555"
095             android:paddingTop="3dip"
096             android:paddingBottom="3dip" />
097         <!-- 数据目录==内容 -->
098         <TextView
099             android:id="@+id/detail_process_data_dir"
100             android:layout_height="wrap_content"
101             android:layout_width="fill_parent"
102             android:paddingTop="3dip"
103             android:paddingBottom="3dip" />
104         <!-- 包的大小==标题 -->
105         <TextView
106             android:text="@string/detail_process_package_size"
107             android:textSize="7pt"
108             android:layout_alignParentLeft="true"
109             android:layout_width="fill_parent"
110             android:layout_height="25dip"
111             android:gravity="center_vertical"
112             android:background="#555555"
113             android:paddingTop="3dip"
114             android:paddingBottom="3dip" />
115         <!-- 包的大小==内容 -->
116         <TextView
117             android:id="@+id/detail_process_package_size"
118             android:layout_height="wrap_content"
119             android:layout_width="fill_parent"
120             android:paddingTop="3dip"
121             android:paddingBottom="3dip" />
122         <!-- 程序拥有的权限==标题 -->
123         <TextView
124             android:text="@string/detail_process_permission"
125             android:textSize="7pt"
126             android:layout_alignParentLeft="true"
127             android:layout_width="fill_parent"
128             android:layout_height="25dip"
129             android:gravity="center_vertical"
130             android:background="#555555"
131             android:paddingTop="3dip"
132             android:paddingBottom="3dip" />
133         <!-- 程序拥有的权限==内容 -->
134         <TextView
135             android:id="@+id/detail_process_permission"
136             android:layout_height="wrap_content"
137             android:layout_width="fill_parent"
138             android:paddingTop="3dip"
139             android:paddingBottom="3dip" />
140         <!-- 程序包含的服务==标题 -->
```

```
141         <TextView
142             android:text="@string/detail_process_service"
143             android:textSize="7pt"
144             android:layout_alignParentLeft="true"
145             android:layout_width="fill_parent"
146             android:layout_height="25dip"
147             android:gravity="center_vertical"
148             android:background="#555555"
149             android:paddingTop="3dip"
150             android:paddingBottom="3dip" />
151         <!-- 程序包含的服务==内容 -->
152         <TextView
153             android:id="@+id/detail_process_service"
154             android:layout_height="wrap_content"
155             android:layout_width="fill_parent"
156             android:paddingTop="3dip"
157             android:paddingBottom="3dip" />
158         <!-- 程序的 Activity==标题 -->
159         <TextView
160             android:text="@string/detail_process_activity"
161             android:textSize="7pt"
162             android:layout_alignParentLeft="true"
163             android:layout_width="fill_parent"
164             android:layout_height="25dip"
165             android:gravity="center_vertical"
166             android:background="#555555"
167             android:paddingTop="3dip"
168             android:paddingBottom="3dip" />
169         <!-- 程序的 Activity==内容 -->
170         <TextView
171             android:id="@+id/detail_process_activity"
172             android:layout_height="wrap_content"
173             android:layout_width="fill_parent"
174             android:paddingTop="3dip"
175             android:paddingBottom="3dip" />
176     </LinearLayout>
```

9.3　功 能 实 现

如图 9.3 所示，为整个工程的目录结构，可以看到在包 com.guo.taskmanager 下有 6 个程序文件。主界面程序文件为 TaskManagerActivity.java，PackageUtil.java 用于获取程序的一些基本信息，ProgtamUtil.java 实现了一个用于存放列表每一行元素的类，ProcListAdapter.java 新建了一个用于适配 ListView 的 Adapter，DetailProgramUtil.java 是一个用于存放进程详细信息的类，相对应的用于显示进程详细信息的 Activity 为 ProcDetailActivity.java。

9.3.1　初始化变量

程序的开始我们还是先初始化一些变量，首先是至关重要的 packageManager 和 activityManager 用于获取程序的包信息和 Activity 信息。列表 runningProcessList 和 infoList 都是用来存放当前运行程序的信息，前者是通过活动管理器获取，后者将前者的信息提取

封装到我们自己创建的类 ProgramUtil 中，以利于显示。

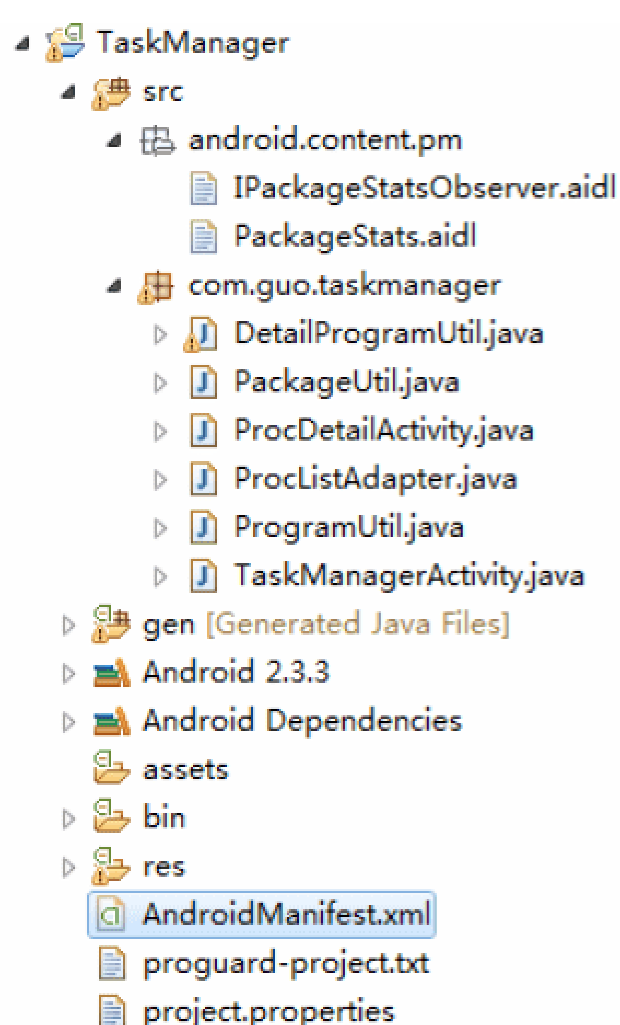


图 9.3　工程结构

processSelected 用于保存当前选中的进程，以进一步对该进程进行操作。此外还有界面的“刷新”和“强制结束”按钮、进度条、列表适配器等。

```
01  //获取包管理器和活动管理器的实例
02  private static PackageManager packageManager = null;
03  private static ActivityManager activityManager = null;
04
05  //正在运行的进程列表
06  private static List<RunningAppProcessInfo> runningProcessList = null;
07  private static List<ProgramUtil> infoList = null;
08  //用于获得应用程序基本信息
09  private static PackageUtil packageUtil = null;
10
11  //被选中的进程名称
12  private static RunningAppProcessInfo processSelected = null;
13  //“刷新”按钮
14  private static Button btnRefresh = null;
15  //“结束进程”按钮
16  private static Button btnCloseAll = null;
17
18  //用于后台刷新列表，显示刷新提示
19  private static RefreshHandler handler = null;
20  //用于显示刷新进度条
21  private static ProgressDialog pd = null;
22  //ListView 的适配器
23  private static ProcListAdapter procListAdapter = null;
```

9.3.2　获取运行的进程

程序运行时，首先执行 onCreate()函数，在该函数中首先初始化界面的两个按钮并绑定监听器，如以下代码 006～009 行所示。接着获得实例化包管理器和活动管理器，然后执行 updateProcessList()函数，更新列表信息。

在 updateProcessList()函数中，显示圆形进度条提示用户正在更新界面，然后执行 RefreshThread 线程。在 RefreshThread 线程中执行了核心函数 buildProcListAdapter()，获得 ListView 的适配器 procListAdapter。

在函数 buildProcListAdapter()中，先将存放当前运行程序信息的两个数组 runningProcessList 和 infoList 清空，然后重新获取当前运行的程序，如代码 066 行所示。接着枚举 runningProcessList 中的元素，并从里面的元素中提取需要的信息存储到类 ProgramUtil 中，最后将得到的数据生成适配器并返回。

在两个存放进程信息的数组进行转换的过程中，buildProgramUtilSimpleInfo 起了关键的作用。通过传入的进程名字可以获得当前应用程序的信息，如代码 088 所示，接着在代码 090～099 行分别设置程序的图标和标签。通过进程的 id 信息，可以获得进程占用的内存，如代码 101 行所示。

```
001    @Override
002    public void onCreate(Bundle savedInstanceState) {
003        super.onCreate(savedInstanceState);
004        setContentView(R.layout.proc_list);
005
006        btnRefresh = (Button)findViewById(R.id.btn_refresh_process);
007        btnRefresh.setOnClickListener(new RefreshButtonListener());
008        btnCloseAll = (Button)findViewById(R.id.btn_closeall_process);
009        btnCloseAll.setOnClickListener(new CloseAllButtonListener());
010
011        //获取包管理器，主要通过包管理器获取程序的图标和程序名
012     packageManager = this.getPackageManager();
013     activityManager = (ActivityManager) getSystemService(ACTIVITY
       _SERVICE);
014     packageUtil = new PackageUtil(this);
015
016     //获取正在运行的进程列表
017     runningProcessList = new ArrayList<RunningAppProcessInfo>();
018     infoList = new ArrayList<ProgramUtil>();
019
020        updateProcessList();
021    }
022 //更新列表信息
023 private void updateProcessList() {
024     //新建一个进度对话框，在刷新列表时，覆盖在父视图之上
025     pd = new ProgressDialog(TaskManagerActivity.this);
026        pd.setProgressStyle(ProgressDialog.STYLE_SPINNER);
027        pd.setTitle(getString(R.string.progress_tips_title));
028        pd.setMessage(getString(R.string.progress_tips_content));
029        //启动新线程，执行更新列表操作
030     handler = new RefreshHandler();
031     RefreshThread thread = new RefreshThread();
032     thread.start();
033     //显示进度对话框
034        pd.show();
035    }
036 private class RefreshHandler extends Handler {
037     @Override
038     public void handleMessage(Message msg) {
039         //更新界面
040         getListView().setAdapter(procListAdapter);
041         //取消进度框
```

```
042         pd.dismiss();
043      }
044    }
045    //用于更新界面的进程
046    class RefreshThread extends Thread {
047     @Override
048     public void run() {
049         //TODO : Do your Stuff here.
050         procListAdapter = buildProcListAdapter();
051         Message msg = handler.obtainMessage();
052         handler.sendMessage(msg);
053      }
054 }
055     //构建一个 ListAdapter
056 public ProcListAdapter buildProcListAdapter() {
057     //清空正在运行的程序
058     if (!runningProcessList.isEmpty()) {
059         runningProcessList.clear();
060     }
061     //清空存放运行程序信息的数组
062     if (!infoList.isEmpty()) {
063         infoList.clear();
064     }
065     //获取正在运行的进程
066     runningProcessList = activityManager.getRunningAppProcesses();
067     RunningAppProcessInfo procInfo = null;
068     for (Iterator<RunningAppProcessInfo> iterator = runningProcessList.
      iterator(); iterator.hasNext();) {
069         procInfo = iterator.next();
070         //将程序的信息存储到类中
071         ProgramUtil programUtil = buildProgramUtilSimpleInfo(procInfo.
            pid, procInfo.processName);
072         //将程序信息添加到数组中
073         infoList.add(programUtil);
074     }
075
076     ProcListAdapter adapter = new ProcListAdapter(infoList, this);
077     return adapter;
078 }
079  /*
080  * 为进程获取基本的信息
081  */
082    public ProgramUtil buildProgramUtilSimpleInfo(int procId, String
       procNameString) {
083
084     ProgramUtil programUtil = new ProgramUtil();
085     programUtil.setProcessName(procNameString);
086
087     //根据进程名，获取应用程序的 ApplicationInfo 对象
088     ApplicationInfo tempAppInfo = packageUtil.getApplicationInfo
      (procNameString);
089
090     if (tempAppInfo != null) {
091         //为进程加载图标和程序名称
092         programUtil.setIcon(tempAppInfo.loadIcon(packageManager));
093         programUtil.setProgramName(tempAppInfo.loadLabel
           (packageManager).toString());
094     }
095     else {
```

```
096         //如果获取失败，则使用默认的图标和程序名
097         programUtil.setIcon(getApplicationContext().getResources().
            getDrawable(R.drawable.unknown));
098         programUtil.setProgramName(procNameString);
099     }
100     //设置进程内存使用量
101     String str = getUsedMemory(procId);
102     programUtil.setMemString(str);
103     return programUtil;
104    }
105     //获得进程占用内存信息
106 public String getUsedMemory(int pid)
107 {
108     //获得活动管理器实例
109     ActivityManager am = (ActivityManager)getSystemService(Context.
        ACTIVITY_SERVICE);
110     //构建 int 数组
111     int[] pids = {pid};
112     MemoryInfo[] memoryInfos =  am.getProcessMemoryInfo(pids);
113     //获得进程占用内存总量，返回 kB 值
114     int memorysize = memoryInfos[0].getTotalPrivateDirty();
115     return "内存占用为 "+ memorysize +" KB";
116 }
```

9.3.3　获取应用程序

如下所示是类 PackageUtil 的实现代码，主要用来获取应用程序的信息，如代码 16 行所示，该程序将会获取所有的应用程序，包括那些已经被卸载但仍然保留数据目录的程序。函数 getApplicationInfo()传入一个应用程序的名称，返回应用程序的信息。

```
01  package com.guo.taskmanager;            //声明包语句
02~05 行为引入相关类，这里不再列举，请阅读光盘内容。
//
……
06  public class PackageUtil {
07      /* ApplicationInfo 类，保存了普通应用程序的信息，
08       *主要是指 Manifest.xml 中 application 标签中的信息
09       **/
10      private List<ApplicationInfo> allAppList;
11      //构造函数
12      public PackageUtil(Context context) {
13          //获得包管理器
14          PackageManager pm = context.getApplicationContext().
            getPackageManager();
15          //获取所有应用程序，包括那些被卸载掉，但仍保留数据目录的程序
16          allAppList = pm.getInstalledApplications(PackageManager.
            GET_UNINSTALLED_PACKAGES);
17      }
18      /**
19       * 通过一个程序名返回该程序的一个 ApplicationInfo 对象
20       * @param name 程序名
21       * @return ApplicationInfo
22       */
23      public ApplicationInfo getApplicationInfo(String appName) {
24          if (appName == null) {
```

```
25              return null;
26          }
27          //遍历，返回对应的应用程序信息
28          for (ApplicationInfo appinfo : allAppList) {
29              if (appName.equals(appinfo.processName)) {
30                  return appinfo;
31              }
32          }
33          return null;
34      }
35  }
```

9.3.4　存放程序的基本信息

下面这个类用来存放程序的基本信息，包括程序的名称、图标、包名和内存占用量。在主界面程序中每次需要存储一个程序的数据时就实例化类 ProgramUtil，调用类中的 set 方法将数据封装到类中。

```
01  package com.guo.taskmanager;                        //声明包语句
02  import android.graphics.drawable.Drawable;          //引入相关类
03
04  public class ProgramUtil{
05      /*
06       * 定义应用程序的简要信息部分
07       */
08      private Drawable icon;                           //程序图标
09      private String programName;                      //程序名称
10      private String processName;
11      private String memString;
12      //初始化变量
13      public ProgramUtil() {
14          icon = null;
15          programName = "";
16          processName = "";
17          memString = "";
18      }
19      //获得图标
20      public Drawable getIcon() {
21          return icon;
22      }
23      //设置图标
24      public void setIcon(Drawable icon) {
25          this.icon = icon;
26      }
27      //获得应用程序名称
28      public String getProgramName() {
29          return programName;
30      }
31      //设置应用程序名称
32      public void setProgramName(String programName) {
33          this.programName = programName;
34      }
35      //获得程序占用内存大小
36      public String getMemString() {
37          return memString;
38      }
```

```
39      //设置程序占用内存大小
40      public void setMemString(String memString) {
41          this.memString = memString;
42      }
43      //获得程序名称
44      public String getProcessName() {
45          return processName;
46      }
47      //设置程序名称
48      public void setProcessName(String processName) {
49          this.processName = processName;
50      }
```

9.3.5　取出信息并适配到 ListView 中

与类 ProgramUtil 相对应，ProcListAdapter 将 ProgramUtil 的信息逐个取出，并适配到 ListView 的每一行中，同样需要适配的数据有进程图标、进程名称、进程包名和进程占用的内存。

```
01  package com.guo.taskmanager;            //声明包语句
02~11 行为引入相关类，这里不再列举，请阅读光盘内容。
//
……
12  public class ProcListAdapter extends BaseAdapter{
13      List<ProgramUtil> list = new ArrayList<ProgramUtil>();
14      //类 LayoutInflater 用于将一个 XML 布局文件实例化为一个 View 对象
15      LayoutInflater layoutInflater;
16      Context context;
17
18      //构造函数，参数为列表对象和 Context
19      public ProcListAdapter(List<ProgramUtil> list, Context context) {
20          this.list = list;
21          this.context = context;
22      }
23      //获得列表元素总数
24      @Override
25      public int getCount() {
26          //TODO Auto-generated method stub
27          return list.size();
28      }
29      //获得列表元素
30      @Override
31      public Object getItem(int position) {
32          //TODO Auto-generated method stub
33          return list.get(position);
34      }
35      //获得列表元素的 id
36      @Override
37      public long getItemId(int position) {
38          //TODO Auto-generated method stub
39          return position;
40      }
41      //获取显示数据的列表的 View 对象
42      @Override
43      public View getView(int position, View convertView, ViewGroup parent){
```

```
44
45          ViewHolder holder = null;
46
47
48          if (convertView == null) {
49              //从给定的 context 中获取 LayoutInflater 对象
50              layoutInflater = LayoutInflater.from(context);
51              //方法 inflate()：从特定的 XML 布局文件 inflate 膨胀出一个新的视图
52              convertView = layoutInflater.inflate(R.layout.proc
                list item, null);
53
54              holder = new ViewHolder();
55              holder.image = (ImageView)convertView.findViewById
                (R.id.image app);
56              holder.nameText = (TextView)convertView.findViewById
                (R.id.name app);
57              holder.processName = (TextView)convertView.findViewById
                (R.id.package app);
58              holder.memInfo = (TextView)convertView.findViewById
                (R.id.cpumem_app);
59
60              //为一个 View 添加标签，标签项在类 ViewHolder 中定义，可以看作是一个
                  绑定操作
61              convertView.setTag(holder);
62          } else {
63              holder = (ViewHolder) convertView.getTag();
64          }
65
66          final ProgramUtil pUtils = list.get(position);
67          //设置图标
68          holder.image.setImageDrawable(pUtils.getIcon());
69          //设置程序名
70          holder.nameText.setText(pUtils.getProgramName());
71          //设置进程名
72          holder.processName.setText(pUtils.getProcessName());
73          //设置内存信息
74          holder.memInfo.setText(pUtils.getMemString());
75
76          return convertView;
77      }
78  }
79  //用于存放每一行所有元素的类
80  class ViewHolder {
81      TextView nameText;
82      TextView processName;
83      TextView memInfo;
84      ImageView image;
```

9.3.6　重写 onListItemClick 方法

当单击主界面的程序时，我们需要弹出一个对话框，如下代码所示，重写 onListItemClick 方法。在该方法中，先获得当前运行的程序并保存到变量 processSelected 中，接着弹出一个定制的对话框。在对话框中设置两个按钮，一个是查看程序详细信息，一个是结束当前进程。

结束当前进程只需直接调用函数 closeOneProgress 并更新界面即可，而要查看程序详

细信息则需要先通过函数 DetailProgramUtil programUtil = buildProgramUtilComplexInfo (processSelected.processName)实例化类 DetailProgramUtil，接着通过 intent 绑定这个类的信息，如代码 31～33 行所示，将数据传递给 ProcDetailActivity 并打开显示程序详细信息的界面。

```
01
02  @Override
03  protected void onListItemClick(ListView l, View v, int position, long
    id) {
04      //获得当前选中的进程
05      processSelected = runningProcessList.get(position);
06      //新建对话框
07      AlertButtonListener listener = new AlertButtonListener();
08      Dialog alertDialog = new AlertDialog.Builder(this)
09          .setIcon(android.R.drawable.ic_dialog_info)
10          .setTitle(R.string.please_select)
11          .setNegativeButton(R.string.kill_process, listener)
12          .setNeutralButton(R.string.info_detail, listener).create();
13      alertDialog.show();
14      super.onListItemClick(l, v, position, id);
15  }
16
17    private class AlertButtonListener implements
18          android.content.DialogInterface.OnClickListener {
19      //按键处理
20      @Override
21      public void onClick(DialogInterface dialog, int which) {
22          switch (which) {
23          case Dialog.BUTTON_NEUTRAL:
24              Intent intent = new Intent();
25              intent.setClass(TaskManagerActivity.this,
                ProcDetailActivity.class);
26              //为选中的进程获取安装包的详细信息
27              DetailProgramUtil programUtil = buildProgramUtilComplexInfo
                (processSelected.processName);
28              if (programUtil == null) {
29                  break;
30              }
31              Bundle bundle = new Bundle();
32              //使用 Serializable 在 Activity 之间传递对象
33              bundle.putSerializable("process_info", (Serializable)
                programUtil);
34              intent.putExtras(bundle);
35              //打开进程详细信息界面
36              startActivity(intent);
37              break;
38          case Dialog.BUTTON_NEGATIVE:
39              //结束进程
40              closeOneProcess(processSelected.processName);
41              //更新界面
42              updateProcessList();
43              break;
44          default:
45              break;
```

```
46          }
47      }
48  }
```

9.3.7　关闭指定进程

关闭指定进程的代码如下所示，首先传入程序的包名称，判断是否是本程序，不是就直接关掉。当然，系统进程是关不掉的。

```
01  //关闭指定进程
02  private void closeOneProcess(String processName) {
03    //阻止用户结束本程序
04    if (processName.equals(R.string.my package)) {
05        Toast.makeText(TaskManagerActivity.this, "Canot Terminate
          Myself!", Toast.LENGTH LONG).show();
06        return;
07    }
08    //通过一个程序名返回该程序的一个 ApplicationInfo 对象
09    ApplicationInfo tempAppInfo = packageUtil.getApplicationInfo
      (processName);
10    if (tempAppInfo == null) {
11        return;
12    }
13    //根据包名关闭进程
14    activityManager.killBackgroundProcesses(tempAppInfo.
       packageName);
15  }
```

9.3.8　显示文件详细信息

关于显示文件详细信息的部分，我们同样需要新建一个类来封装程序的详细信息。大家可能发现了，当我们需要描述一个对象的时候，通常新建一个类，然后在这个类中新建一些属性和方法，这样我们可以很方便地存取信息。信息以一个个对象的方式存在，将会比较便于理解和管理。

从下面代码可以看到，程序的详细信息包含很多属性，包括程序的包名、进程的 ID、进程名、版本代号、版本名称等等。在程序的构造函数中，我们先将这些成员属性进行初始化，如代码 033～046 行所示。接着从 047～176 行都是对这些成员属性进程读取和赋值的方法。133～177 行中是 3 个重载方法，用来将不同类型的数组转换成字符串，其实现方法差不多。第一个函数的参数是字符串数组，直接取出每一个元素，以换行符拼接并返回，第二个函数和第三个函数都是将参数通过 toString 先转化成字符串，再以换行符拼接并返回。

```
001 package com.guo.taskmanager;                  //声明包语句
002~006 行为引入相关类，这里不再列举，请阅读光盘内容。
//
……
007 public class DetailProgramUtil implements Serializable{
008    private static final long serialVersionUID = 1L;
009    /*
010     * 定义应用程序的扩展信息部分
```

```
011      */
012
013     private String packageName;              //包名
014     private int pid;                         //程序的进程 id
015     private String processName;              //程序运行的进程名
016
017     private String companyName;              //公司名称
018     private int versionCode;                 //版本代号
019     private String versionName;              //版本名称
020
021     private String dataDir;                  //程序的数据目录
022     private String sourceDir;                //程序包的源目录
023
024     private String firstInstallTime;         //第一次安装的时间
025     private String lastUpdateTime;           //最近的更新时间
026
027     private String userPermissions;          //应用程序的权限
028     private String activities;               //应用程序包含的 Activities
029     private String services;                 //应用程序包含的服务
030
031     //android.content.pm.PackageState 类的包信息
032     //构造函数，初始化数据
033     public DetailProgramUtil() {
034         pid = 0;
035         processName = "";
036         companyName = "";
037         versionCode = 0;
038         versionName = "";
039         dataDir = "";
040         sourceDir = "";
041         firstInstallTime = "";
042         lastUpdateTime = "";
043         userPermissions = "";
044         activities = "";
045         services = "";
046     }
047     //获得进程 id
048     public int getPid() {
049         return pid;
050     }
051     //设置进程 id
052     public void setPid(int pid) {
053         this.pid = pid;
054     }
055     //获得版本号
056     public int getVersionCode() {
057         return versionCode;
058     }
059     //设置版本号
060     public void setVersionCode(int versionCode) {
061         this.versionCode = versionCode;
062     }
063     //获得版本信息
064     public String getVersionName() {
065         return versionName;
066     }
067     //设置版本信息
068     public void setVersionName(String versionName) {
```

```
069         this.versionName = versionName;
070     }
071     //获得公司名称
072     public String getCompanyName() {
073         return companyName;
074     }
075     //设置公司名称
076     public void setCompanyName(String companyString) {
077         this.companyName = companyString;
078     }
079     //获得初次安装时间
080     public String getFirstInstallTime() {
081         if (firstInstallTime == null || firstInstallTime.length() <= 0){
082             firstInstallTime = "null";
083         }
084         return firstInstallTime;
085     }
086     //设置初次安装时间
087     public void setFirstInstallTime(long firstInstallTime) {
088         this.firstInstallTime = DateFormat.format(
089                 "yyyy-MM-dd", firstInstallTime).toString();
090     }
091     //获得最后更新时间
092     public String getLastUpdateTime() {
093         if (lastUpdateTime == null || lastUpdateTime.length() <= 0) {
094             lastUpdateTime = "null";
095         }
096         return lastUpdateTime;
097     }
098     //设置最后更新时间
099     public void setLastUpdateTime(long lastUpdateTime) {
100         this.lastUpdateTime = DateFormat.format(
101                 "yyyy-MM-dd", lastUpdateTime).toString();
102     }
103     //获得活动
104     public String getActivities() {
105         if (activities == null || activities.length() <= 0) {
106             activities = "null";
107         }
108         return activities;
109     }
110     //设置活动
111     public void setActivities(ActivityInfo[] activities) {
112         this.activities = Array2String(activities);
113     }
114     //获得用户权限
115     public String getUserPermissions() {
116         if (userPermissions == null || userPermissions.length() <= 0) {
117             userPermissions = "null";
118         }
119         return userPermissions;
120     }
121     //设置用户权限
122     public void setUserPermissions(String[] userPermissions) {
123         this.userPermissions = Array2String(userPermissions);
124     }
125     //获得服务
126     public String getServices() {
```

```
127         if (services == null || services.length() <= 0) {
128             services = "null";
129         }
130         return services;
131     }
132     //设置服务
133     public void setServices(ServiceInfo[] services) {
134         this.services = Array2String(services);
135     }
136     //获得进程名称
137     public String getProcessName() {
138         if (processName == null || processName.length() <= 0) {
139             processName = "null";
140         }
141         return processName;
142     }
143     //设置进程名称
144     public void setProcessName(String processName) {
145         this.processName = processName;
146     }
147     //获得数据目录
148     public String getDataDir() {
149         if (dataDir == null || dataDir.length() <= 0) {
150             dataDir = "null";
151         }
152         return dataDir;
153     }
154     //设置数据目录
155     public void setDataDir(String dataDir) {
156         this.dataDir = dataDir;
157     }
158     //获得缓存的目录
159     public String getSourceDir() {
160         if (sourceDir == null || sourceDir.length() <= 0) {
161             sourceDir = "null";
162         }
163         return sourceDir;
164     }
165     //设置源路径
166     public void setSourceDir(String sourceDir) {
167         this.sourceDir = sourceDir;
168     }
169     //设置包名
170     public void setPackageName(String packageName) {
171         this.packageName = packageName;
172     }
173     //获得包名
174     public String getPackageName() {
175         return packageName;
176     }
177     /*
178      * 3 个重载方法，参数不同，调用不同的方法，用于将 object 数组转化成要求的字符串
179      */
180     //用户权限信息
181     public String Array2String(String[] array) {
182 
183         String resultString = "";
184         if (array != null && array.length > 0) {
185             resultString = "";
```

```
186             //遍历数组，使用换行符拼接数据
187             for (int i = 0; i < array.length; i++) {
188                 resultString += array[i];
189                 if (i < (array.length - 1)) {
190                     resultString += "\n";
191                 }
192             }
193         }
194         return resultString;
195     }
196 
197     //服务信息
198     public String Array2String(ServiceInfo[] array) {
199         String resultString = "";
200         //遍历数组，使用换行符拼接数据
201         if (array != null && array.length > 0) {
202             resultString = "";
203             for (int i = 0; i < array.length; i++) {
204                 if (array[i].name == null) {
205                     continue;
206                 }
207                 resultString += array[i].name.toString();
208                 if (i < (array.length - 1)) {
209                     resultString += "\n";
210                 }
211             }
212         }
213         return resultString;
214     }
215 
216     //活动信息
217     public String Array2String(ActivityInfo[] array) {
218         String resultString = "";
219         //遍历数组，使用换行符拼接数据
220         if (array != null && array.length > 0) {
221             resultString = "";
222             for (int i = 0; i < array.length; i++) {
223                 if (array[i].name == null) {
224                     continue;
225                 }
226                 resultString += array[i].name.toString();
227                 if (i < (array.length - 1)) {
228                     resultString += "\n";
229                 }
230             }
231         }
232         return resultString;
233     }
```

9.3.9　显示程序详细信息

以下就是用于显示程序详细信息的 Activity，整个程序界面的显示都在 onCreate 中完成。在 onCreate 中先是获得程序的界面元素，并为唯一的按钮——“强制结束”绑定监听器，取得传递过来的经过序列化的变量，接着调用核心函数 showAppInfo()将信息显示到界面。

因为前面已经将大部分信息都存放到 processInfo 变量中了，因此只需调用 setText()函

数将 processInfo 变量逐一显示到界面即可。为什么说是大部分呢？因为我们还没有获得程序的包大小信息，这个看似简单的事情其实包含了很多知识点。如果我们直接调用函数去获取包大小信息的话将会失败，因为这些信息是程序的私有变量，而且没有对外提供获取的方法。怎么办呢？在这里我们可以通过 AIDL 和程序反射机制获得所要的信息。

如代码 127～138 行所示，我们使用反射机制获得程序的私有方法 getPackageSizeInfo，并执行。再通过 AIDL 获得程序的执行结果，即我们要的程序包大小信息 pStats，通过消息机制传递给 mHandler，如代码 141～150 行所示。最后将 mHandler 获得的信息，即程序的包大小信息设置到界面相应位置中。

在显示程序大小信息时候我们用到了函数 formatFileSize，将数字进行一次格式化，根据数字的大小转换成 GB、MB、KB、B。

```
001 package com.guo.taskmanager;             //声明包语句
002~018 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
019 //程序详细信息界面
020 public class ProcDetailActivity extends Activity {
021     private static final String ATTR PACKAGE STATS="PackageStats";
022     private DetailProgramUtil processInfo = null;
023     private static TextView textProcessName = null;
024     private static TextView textProcessVersion = null;
025     private static TextView textInstallDir = null;
026     private static TextView textDataDir = null;
027     private static TextView textPkgSize = null;
028     private static TextView textPermission = null;
029     private static TextView textService = null;
030     private static TextView textActivity = null;
031     private static Button btnKillProcess = null;
032
033     @Override
034     protected void onCreate(Bundle savedInstanceState) {
035         //TODO Auto-generated method stub
036         super.onCreate(savedInstanceState);
037         setContentView(R.layout.proc detail);
038         //获得界面元素
039         //程序名称
040         textProcessName = (TextView)findViewById(R.id.detail process
            name);
041         //程序版本
042         textProcessVersion = (TextView)findViewById(R.id.detail_
            process_copyright);
043         //安装目录
044         textInstallDir = (TextView)findViewById(R.id.detail process
            install dir);
045         //数据目录
046         textDataDir = (TextView)findViewById(R.id.detail process
            data_dir);
047         //包的大小信息
048         textPkgSize = (TextView)findViewById(R.id.detail_process_
            package size);
049         //权限信息
050         textPermission = (TextView)findViewById(R.id.detail process
            permission);
051         //服务信息
```

```
052         textService = (TextView)findViewById(R.id.detail_process_
            service);
053         //Activity 信息
054         textActivity = (TextView)findViewById(R.id.detail_process_
            activity);
055         //"强制结束"按钮
056         btnKillProcess = (Button)findViewById(R.id.btn_kill_process);
057         //绑定监听器
058         btnKillProcess.setOnClickListener(new KillButtonListener());
059         //获得传递过来的数据
060         Intent intent = getIntent();
061         Bundle bundle= intent.getExtras();
062         processInfo = (DetailProgramUtil) bundle.
            getSerializable("process_info");
063         //将获得的数据显示到界面
064         showAppInfo();
065     }
066     //显示程序详细信息
067     public void showAppInfo(){
068         //设置程序名称
069         textProcessName.setText(processInfo.getProcessName());
070         //设置程序安装目录
071         textInstallDir.setText(processInfo.getSourceDir());
072         //设置程序版本
073         textProcessVersion.setText(
074                 getString(R.string.detail_process_company) +
                    processInfo.getCompanyName()
075                 + "  " + getString(R.string.detail_process_version) +
                    processInfo.getVersionName()
076                 + "(" + processInfo.getVersionCode() + ")");
077         //设置数据目录
078         textDataDir.setText(processInfo.getDataDir());
079         //设置权限
080         textPermission.setText(processInfo.getUserPermissions());
081         //设置服务信息
082         textService.setText(processInfo.getServices());
083         //设置 Activity 信息
084         textActivity.setText(processInfo.getActivities());
085         //设置包大小信息
086         getpkginfo(processInfo.getPackageName());
087     }
088     // “结束进程”按钮监听器
089     private class KillButtonListener implements OnClickListener {
090         @Override
091         public void onClick(View v) {
092             //获得活动管理器
093             ActivityManager activityManager = (ActivityManager)
                getSystemService(ACTIVITY_SERVICE);
094             PackageUtil packageUtil = new PackageUtil(ProcDetailActivity.
                this);
095             //如果是本程序，则不结束
096             if (processInfo.getProcessName().equals(R.string.
                my_package)) {
097                 Toast.makeText(ProcDetailActivity.this, "Canot
                    Terminate Myself!", Toast.LENGTH_LONG).show();
098                 return;
099             }
100             //获得程序的信息类
```

```
101             ApplicationInfo tempAppInfo = packageUtil
               .getApplicationInfo(processInfo.getProcessName());
102             activityManager.killBackgroundProcesses(tempAppInfo.
                packageName);
103             Toast.makeText(ProcDetailActivity.this, "Process is
                killed!", Toast.LENGTH LONG).show();
104         }
105     }
106         //用于更新界面的报数据大小信息
107      private Handler mHandler = new Handler() {
108             public void handleMessage(Message msg) {
109                 switch (msg.what) {
110                 case 1:
111                     String infoString="";
112                     PackageStats newPs = msg.getData().getParcelable
                        (ATTR PACKAGE STATS);
113                     if (newPs!=null) {
114                         infoString+="应用程序大小: "+formatFileSize(newPs.
                            codeSize);
115                         infoString+="\n 数据大小: "+formatFileSize(newPs.
                            dataSize);
116                         infoString+="\n 缓存大小: "+formatFileSize(newPs.
                            cacheSize);
117                     }
118                     textPkgSize.setText(infoString);
119                     break;
120                 default:
121                     break;
122                 }
123             }
124             };
125     //利用反射机制获得程序的包大小信息
126     //使用普通方式将得不到包的大小信息
127     public void getpkginfo(String pkg){
128         PackageManager pm = getPackageManager();
129         try {
130             Method getPackageSizeInfo = pm.getClass()
131             .getMethod("getPackageSizeInfo", String.class,
132                     IPackageStatsObserver.class);
133             getPackageSizeInfo.invoke(pm, pkg,
134             new PkgSizeObserver());
135         } catch (Exception e) {
136             //TODO: handle exception
137         }
138     }
139     //使用 AIDL 实现进程间通信，将包的信息发送给 mHandler
140     class PkgSizeObserver extends IPackageStatsObserver.Stub {
141   public void onGetStatsCompleted(PackageStats pStats, boolean
      succeeded) {
142        Message msg = mHandler.obtainMessage(1);
143        Bundle data = new Bundle();
144        //将包信息存放到 data 中
145        data.putParcelable(ATTR PACKAGE STATS, pStats);
146        msg.setData(data);
147        mHandler.sendMessage(msg);
148
149    }
150 }
151      //格式化文件大小信息
152     public static String formatFileSize(long length) {
```

```
153         String result = null;
154         int sub string = 0;
155         //文件是 GB 级别的情况
156         if (length >= 1073741824) {
157             sub_string=String.valueOf((float) length/1073741824).
                indexOf(
158                     ".");
159             result = ((float) length/1073741824 + "000").substring(0,
160                     sub_string + 3)
161                     + "GB";
162             //文件是 MB 级别的情况
163         } else if (length >= 1048576) {
164             sub string=String.valueOf((float) length/1048576).
                indexOf(".");
165             result = ((float) length/1048576 + "000").substring(0,
166                     sub string + 3)
167                     + "MB";
168             //文件是 KB 级别的情况
169         } else if (length >= 1024) {
170             sub string=String.valueOf((float) length/1024)
               .indexOf(".");
171             result=((float) length/1024 + "000").substring(0,
172                     sub string + 3)
173                     + "KB";
174             //文件是 B 级别的情况
175         } else if (length < 1024)
176             result = Long.toString(length) + "B";
177         return result;
178     }
```

9.3.10　更新列表

最后，我们再回到主界面看看底下两个按钮如何实现，“刷新”按钮直接调用函数 updateProcessList()，而“全部结束”则遍历当前的进程，逐个关闭。当然对于系统进程，这种做法是无效的，最终结果是关闭了除当前程序外的所有用户程序。

```
01    //更新列表
02    private class RefreshButtonListener implements android.view.View.
      OnClickListener {
03      @Override
04      public void onClick(View v) {
05          updateProcessList();
06      }
07    }
08    //关闭所有用户程序
09    private class CloseAllButtonListener implements android.view
      .View.OnClickListener {
10      @Override
11      public void onClick(View v) {
12          int count = infoList.size();
13          ProgramUtil bpu = null;
14          //遍历所有进程，逐个关闭
15          for (int i = 0; i < count; i++) {
16              bpu = infoList.get(i);
17              closeOneProcess(bpu.getProcessName());
```

```
18          }
19          //更新列表
20          updateProcessList();
21      }
22    }
```

9.3.11　查看程序详细信息

最后我们需要在 AndroidManifest.xml 中声明权限，查看程序详细信息的代码如下，效果如图 9.4 所示。

```
1   <uses-permission android:name="android.permission.GET_TASKS" />
2   <uses-permission android:name="android.permission.KILL_BACKGROUND_
    PROCESSES" />
3   <uses-permission android:name="android.permission.GET_PACKAGE_SIZE"/>
```



图 9.4　程序详细信息

9.4　知 识 拓 展

在文章的最后我们用到了 Java 的反射机制，那么什么是反射呢？

反射主要是指程序可以访问、检测和修改它本身状态或行为的一种能力。在计算机科学领域，反射是一类应用，它们能够自描述和自控制。这类应用通过某种机制来实现对自

己行为的描述和检测，并能根据自身行为的状态和结果，调整或修改应用所描述行为的状态和相关的语义。

在 Java 中的反射机制，被称为 Reflection。它允许运行中的 Java 程序对自身进行检查，并能直接操作程序的内部属性或方法。Reflection 机制允许程序在执行的过程中，利用 Reflection APIs 取得任何已知名称的类的内部信息，包括 package、type parameters、superclass、implemented interfaces、inner classes、outer classes、fields、constructors、methods、modifiers 等，并可以在执行的过程中，动态生成 Instances、变更 fields 内容或唤起 methods。

因此我们可以利用反射机制，在 Java 程序中动态地调用一些 protected 甚至是 private 的方法或类，这样可以很大程度上满足一些比较特殊的需求。那么反射机制在 Android 平台下有何用处呢？

如果你看过 Android 的源码，会发现很多类或方法中经常加上了“@hide”注释标记，它的作用是使这个方法或类在生成 SDK 时不可见，这就导致我们的程序不能使用这些类或方法。对于这个问题有一种解决方法就是使用 Java 反射机制，利用这种反射机制访问存在访问权限的方法或属性。

下面我们还是来看一个小例子，进一步了解一下反射机制，这个例子是为了实现当单击 AlertDialog 的按钮时不会关掉对话框。

当我们调用系统的 AlertDialog 显示对话框时，不论你如何设置按钮的功能，当单击了按钮时都会关闭对话框。要了解这其中的缘由，我们需要分析一下 SDK 中的代码。

进入 AlertDialog 类的源代码，在 AlertDialog 中只定义了一个变量：mAlert，这个变量是 AlertController 类型。AlertController 类是 Android 的内部类，在 com.android.internal.app 包中，无法通过普通的方式访问，但可以直接在 Android 源代码中找到 AlertController.java。

找到 AlertController.java 文件，找到控制按钮的代码，如下所示可以看出不管是哪个按钮，最终都执行了 16 行的代码，通过代码我们可以猜出就是这句发送了关闭对话框的消息。

```
01  View.OnClickListener mButtonHandler  =  new  View.OnClickListener() {
02        public   void  onClick(View v)  {
03           Message m  =  null ;
04            if  (v  ==  mButtonPositive  &&  mButtonPositiveMessage  !=
             null ) {
05              m  =  Message.obtain(mButtonPositiveMessage);
06           } else   if  (v  ==  mButtonNegative  &&  mButtonNegati-
           veMessage  !=   null ) {
07              m  =  Message.obtain(mButtonNegativeMessage);
08           } else   if  (v  ==  mButtonNeutral  &&  mButtonNeutral-
           Message  !=   null ) {
09              m  =  Message.obtain(mButtonNeutralMessage);
10           }
11            if  (m  !=   null ) {
12              m.sendToTarget();
13           }
14
15            //Post a message so we dismiss after the above handlers are
            executed
16           mHandler.obtainMessage(ButtonHandler.MSG_DISMISS_DIALOG,
           mDialogInterface)
17                  .sendToTarget();
18       }
19    };
```

进一步追踪代码，如代码 21 行、22 行所示，果然我们猜想的没错，程序在这里执行了关闭对话框的操作。

```
01 private  static  final  class ButtonHandler extends Handler {
02       //Button clicks have Message.what as the BUTTON{1,2,3} constant
03       private  static  final  int  MSG DISMISS DIALOG  =  1 ;
04
05       private  WeakReference < DialogInterface >  mDialog;
06
07       public  ButtonHandler(DialogInterface dialog) {
08          mDialog  =  new  WeakReference < DialogInterface > (dialog);
09       }
10
11      @Override
12       public   void  handleMessage(Message msg) {
13           switch  (msg.what) {
14
15               case  DialogInterface.BUTTON POSITIVE:
16               case  DialogInterface.BUTTON NEGATIVE:
17               case  DialogInterface.BUTTON NEUTRAL:
18                  ((DialogInterface.OnClickListener) msg.obj).onClick
                    (mDialog.get(), msg.what);
19                  break ;
20
21               case  MSG DISMISS DIALOG:
22                  ((DialogInterface) msg.obj).dismiss();
23           }
24       }
25   }
```

从上面的分析，我们可以知道，只要我们使用自己的 Handler 对象替换 ButtonHandler，就可以阻止调用 dismiss 方法来关闭对话框，也就是替换掉 mHandler。

下面是程序的代码，在代码32～42行中我们实现了反射，并用我们自定义的MyHandler替换掉了源码中的 ButtonHandler，使得我们有机会重新定义 AlertDialog 的处理机制。在 MyHandler 中，我们模仿 ButtonHandler 的写法，唯一不同的是我们去掉了隐藏对话框的操作。最后，我们看一下运行效果，如图 9.5 所示，单击“关闭”按钮，对话框将不会消失。

```
01 package com.guo.alertdialog;             //声明包语句
02~16 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
17 public class AlertDialogReflactionActivity extends Activity {
18     //对话框
19      AlertDialog alertDialog = null;
20     /** Called when the activity is first created. */
21     @Override
22     public void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         setContentView(R.layout.main);
25         //新建一个对话框
26         Builder mBuilder=new AlertDialog.Builder(this);
```

```
27            mBuilder.setMessage("Hello!");
28            mBuilder.setTitle("提示! ");
29            mBuilder.setNegativeButton("关闭", new myListener());
30            mBuilder.setPositiveButton("确定", new myListener());
31            alertDialog= mBuilder.create();
32            try {
33                //获得 mAlert 私有变量
34                Field mfield=alertDialog.getClass().getDeclaredField
                  ("mAlert");
35                mfield.setAccessible(true);
36                //获得 mAlert 变量在 alertDialog 的值
37                Object obj=mfield.get(alertDialog);
38                mfield=obj.getClass().getDeclaredField("mHandler");
39                //设置是否检查使用权限, true 表示不检查
40                mfield.setAccessible(true);
41                //设置特定的 obj 中 mfield 的值
42                mfield.set(obj, new MyHandler(alertDialog));
43            } catch (Exception e) {
44                e.printStackTrace();
45            }
46            alertDialog.show();
47        }
48        class myListener implements DialogInterface.OnClickListener{
49            @Override
50            public void onClick(DialogInterface dialog, int which) {
51                //TODO Auto-generated method stub
52                switch(which)
53                {
54                //"确定"按钮
55                    case DialogInterface.BUTTON POSITIVE:
56                        dialog.dismiss();
57                        break;
58                //"取消"按钮
59                    case DialogInterface.BUTTON NEGATIVE:
60                        break;
61                }
62            }
63        }
64        //自定义的处理器
65        private class MyHandler extends Handler{
66        private static final int MSG DISMISS DIALOG = 1;
67        private WeakReference<DialogInterface> mDialog;
68        public MyHandler(DialogInterface dialog){
69            mDialog=new WeakReference<DialogInterface>(dialog);
70        }
71        @Override
72        public void handleMessage(Message msg) {
73            switch (msg.what) {
74                case DialogInterface.BUTTON POSITIVE:
75                case DialogInterface.BUTTON NEGATIVE:
76                case DialogInterface.BUTTON NEUTRAL:
77                     ((DialogInterface.OnClickListener)
```

```
                    msg.obj).onClick(mDialog.get(), msg.what);
78                  break;
79                  //此处去掉了隐藏对话框的操作
80              case MSG_DISMISS_DIALOG:
81                  break;
82              }
83          }
84      };
```



图 9.5　对话框

9.5　本章小结

本章主要介绍了如何开发一款任务管理器，在任务管理器中实现查看当前所有进程的状态。通过这个任务管理器，可以查看进程的图标、名称、包名和占用内存的情况。进一步，单击任意程序还可以查看进程的详细信息，包括包数据的大小、安装路径、拥有的权限等。还有，用户可以结束掉任意一个用户进程。在文章的最后，我们介绍了 Android 反射机制的应用。

第 10 章　软件管理器

软件管理器是什么呢？相信大家都用过 Windows 下控制面板中的程序卸载功能，是的，你们猜对了，我们本章要做的就是实现这样功能的程序管理器。

10.1　功 能 分 析

软件管理器，顾名思义就是一个可以对安装在本机的软件进行集中管理的一个软件，我们可以通过它查看已安装的软件，可以查看软件的详细信息，启动和卸载软件等。首先我们还是先看一下最终的效果图，如图 10.1 所示。

程序的左下角有两个图标，左边是视图切换按钮，可以在网格视图和列表视图之间切换，右边是用户程序和所有程序的切换按钮。此外当我们单击软件管理器里面的任意一个软件时将会弹出对话框，如图 10.2 所示，可以查看软件的详细信息，可以启动程序和卸载程序。



图 10.1　软件管理器



图 10.2　管理应用程序

10.2　界 面 设 计

10.1 节介绍了软件管理器的常见功能，本节就来讲解下如何实现这些功能，具体如下。

10.2.1　主界面的设置

如下代码所示，为主界面 main.xml 的代码。主界面可以分为三部分，第一部分是最上面的标题栏，第二部分是中间用来显示程序信息的主体部分，第三部分是最下面的切换图标。

```
001 <?xml version="1.0" encoding="utf-8"?>
002 <RelativeLayout
003     xmlns:android="http://schemas.android.com/apk/res/android"
004     android:layout_width="fill_parent"
005     android:layout_height="fill_parent"
006     android:background="#313849">
007
008     <!--LinearLayout
009         a) 高度为 28px
010         b) 布局方向为水平布局
011         c) 子控件的对齐方式：垂直居中
012         d) 左边预留 5px 的空间
013         e) 设置背景图片为 top_bg
014         f) 包含两个控件
015             i. ImageView
016                 1. 宽 18px
017                 2. 高 18px
018                 3. 图片：manage
019             ii. TextView
020                 1. 高：wrap_content
021                 2. 高：wrap_content
022                 3. 颜色：#000
023                 4. 大小：14px
024                 5. 显示文字为:应用程序 -->
025     <LinearLayout
026         android:layout_height="28px"
027         android:layout_width="fill_parent"
028         android:orientation="horizontal"
029         android:gravity="center_vertical"
030         android:paddingLeft="5px"
031         android:background="@drawable/top_bg">
032         <ImageView
033             android:layout_width="18px"
034             android:layout_height="18px"
035             android:src="@drawable/manage"/>
036         <TextView
037             android:layout_width="wrap_content"
038             android:layout_height="wrap_content"
039             android:textColor="#000"
040             android:textSize="14px"
041             android:text="应用程序"/>
```

```
042     </LinearLayout>
043     <!--GridView
044         a) 配置 id 为:gv apps
045         b) 高、宽: fill_parent
046         c) 列数: 3
047         d) Item 之间的水平间隔和垂直间隔都是 10px
048         e) 当 item 选中时显示的图片为 choose_gridview
049         f) 距离父窗体的上、下、左、右距离分别为 28px、58px、5px、5px -->
050     <GridView
051         android:id="@+id/gridView"
052         android:layout_height="fill_parent"
053         android:layout_width="fill_parent"
054         android:numColumns="3"
055         android:horizontalSpacing="10px"
056         android:verticalSpacing="10px"
057         android:listSelector="@drawable/choose_gridview"
058         android:layout_marginTop="28px"
059         android:layout_marginBottom="58px"
060         android:layout marginLeft="5px"
061         android:layout_marginRight="5px"/>
062    <!--
063     配置 id 为:lv apps
064     高、宽: fill_parent
065     当 item 选中时显示的图片为 choose_gridview
066     距离父窗体的上、下、左、右距离分别为 28px、58px、5px、5px
067     默认为不可见
068     -->
069     <ListView
070         android:id="@+id/lv apps"
071         android:layout width="fill parent"
072         android:layout height="fill parent"
073         android:listSelector="@drawable/choose listview"
074         android:layout marginTop="28px"
075         android:layout marginBottom="58px"
076         android:layout marginLeft="5px"
077         android:layout marginRight="5px"
078         android:visibility="gone" />
079 <!--
080     setVisibility(View.INVISIBLE);只是看不到,但位置仍然占用
081     setVisibility(View.GONE);不仅看不到了,而且连占用的空间也释放了
082
083  -->
084
085     <!--RelativeLayout:
086         a) 宽: fill_parent
087         b) 高: 58px
088         c) 紧靠父控件底部
089         d)背景图片: bottom_bg
090         e) 有两个 ImageButton
091             i. 设置 id 分别为: ib_change_view 和 ib_change_category
092             ii. 一个紧靠左边, 一个紧靠右边
093             iii. 距离父窗体: 5px、1px
094             iv. 图片分别为 list 和 all
095             v. 宽高分别为: 76px 和 54px -->
096
097     <RelativeLayout
098         android:layout width="fill parent"
099         android:layout_height="56px"
```

```
100         android:layout alignParentBottom="true"
101         android:background="@drawable/bottom bg">
102         <!--  -->
103         <ImageButton
104             android:id="@+id/ib change view"
105             android:layout alignParentLeft="true"
106             android:layout marginLeft="5px"
107             android:layout marginTop="5px"
108             android:src="@drawable/list"
109             android:layout width="76px"
110             android:layout height="52px"/>
111         <!-- 用户程序和所有程序切换按钮 -->
112         <ImageButton
113             android:id="@+id/ib_change_category"
114             android:layout width="76px"
115             android:layout height="52px"
116             android:layout alignParentBottom="true"
117             android:layout toRightOf="@+id/ib change view"
118             android:src="@drawable/all" />
119     </RelativeLayout>
120 </RelativeLayout>
```

10.2.2　设置 ListView 布局

接下来是 ListView 的元素布局，代码如下所示，由一个 ImageView 和两个 TextView 组成。ImageView 放置于左边，右边两个 TextView 上下相叠，分别是应用程序名和包名。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <!-- 整体是一个 LinearLayout
03      1、 布局方向为水平布局
04      2、 宽: fill_parent
05      3、 高: wrap_content
06      4、 内容的方向为垂直居中
07      5、 ImageView
08          a) Id 为 lv_icon
09          b) 距离父控件的顶部和底部都是 5px
10          c) 宽、高: 48px
11      6、 LinearLayout
12          a) 布局方向为垂直布局
13          b) 宽: wrap_content
14          c) 高: 48px
15          d) 左边预留 5px 的距离
16          e) TextView
17              i. Id 为 lv item appname
18              ii. 宽: fill_parent
19              iii. 高: wrap_content
20              iv. 单行显示
21              v. 字体大小: 16px
22              vi. 字体样式:加粗
23              vii. 字体颜色:#fff
24          f) TextView
25              i. Id 为 lv_item_packagenam
26              ii. 宽: fill parent
27              iii. 高: wrap_content
28              iv. 单行显示
```

```
29              v. 字体颜色#fff
30  -->
31  <LinearLayout
32      xmlns:android="http://schemas.android.com/apk/res/android"
33      android:orientation="horizontal"
34      android:layout_width="fill_parent"
35      android:layout_height="wrap_content"
36      android:gravity="center_vertical">
37
38      <ImageView
39          android:id="@+id/lv_icon"
40          android:layout_width="48px"
41          android:layout_height="48px"
42          android:layout_marginTop="5px"
43          android:layout_marginBottom="5px" />
44      <LinearLayout
45          android:orientation="vertical"
46          android:layout_width="wrap_content"
47          android:layout_height="48px"
48          android:paddingLeft="5px">
49          <TextView
50              android:id="@+id/lv_item_appname"
51              android:layout_width="fill_parent"
52              android:layout_height="wrap_content"
53              android:singleLine="true"
54              android:textSize="16px"
55              android:textStyle="bold"
56              android:textColor="#fff" />
57          <TextView
58              android:id="@+id/lv_item_packagename"
59              android:layout_width="fill_parent"
60              android:layout_height="wrap_content"
61              android:singleLine="true"
62              android:textColor="#fff" />
63      </LinearLayout>
```

10.2.3　设置 GridView 的子元素布局

用于存放程序信息的另一种形式 GridView 的子元素布局代码如下所示，上面是一个 ImageView，用于显示程序的 icon，下面是一个 TextView，用于显示程序的名称。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <!-- LinearLayout
03      i. 高：wrap_content
04      ii. 宽：90px
05      iii. 布局方向：垂直
06      iv. 设置里面的控件的位置为中间
07      v. ImageView
08          1. 设置 id 为 gv_item_icon
09          2. 宽、高都为 64px
10      vi. TextView
11          1. 设置 id 为 gv_item_appname
12          2. 宽、高都为 wrap_content
13          3. 设置为 2 行
14          4. 字体大小：16px
15          5. 颜色：#FFF -->
16  <LinearLayout
```

```
17      xmlns:android="http://schemas.android.com/apk/res/android"
18      android:orientation="vertical"
19      android:layout width="90px"
20      android:layout height="wrap content"
21      android:gravity="center">
22
23      <ImageView
24         android:id="@+id/gv item icon"
25         android:layout width="64px"
26         android:layout height="64px"/>
27      <TextView
28         android:id="@+id/gv_item_appname"
29         android:layout_width="wrap_content"
30         android:layout_height="wrap_content"
31         android:lines="2"
32         android:textSize="16px"
33         android:textColor="#FFF" />
34   </LinearLayout>
```

10.3　功 能 实 现

整个程序的设计思路很简单，只需要一个界面，在这个界面通过控制 setVisibility 方法来控制显示 GridView 视图还是 ListView 视图，通过改变 Adapter 的数据来控制显示所有应用程序还是用户的应用程序。整个工程的结构如图 10.3 所示。

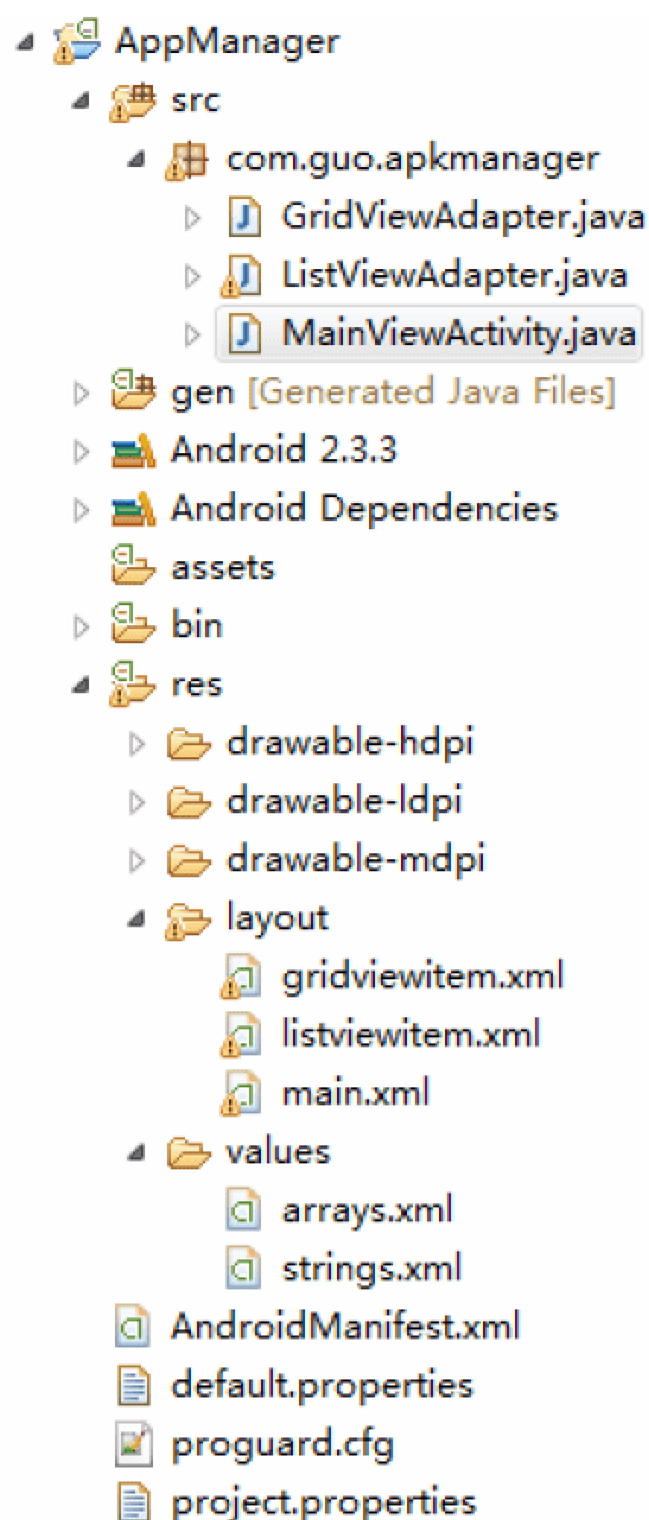


图 10.3　AppManager 工程结构

从图 10.3 可以看出，我们程序的主体文件是 MainViewActivity.java，在该文件中我们

新建一个 Activity，继承于 Activity 并实现 Runnable 的接口，如下所示：

```
public class MainViewActivity extends Activity implements Runnable {
```

10.3.1　声明变量

在程序的开始，我们先声明一些重要的变量，如下所示。变量 packageInfo 用来存放系统中所有程序的信息，而 userPackagetInfos 用来存放用户应用程序的信息，同时为两个切换按钮分别设置标志变量 isUserApp 和 isListView。

```
01  private GridView gridView = null;
02  //用来取得系统中所有包的信息
03  private List<PackageInfo> packageInfos = null;
04  private ImageButton changeCategoryBtn = null;
05  //用户自己安装的程序的信息
06  private List<PackageInfo> userPackageInfos = null;
07  //用来实现系统应用与自己应用的切换
08  private boolean isUserApp = true;
09  private ListView listView = null;
10  private ImageButton changeViewBtn = null;
11  //用来实现 ListView、GridView 的切换
12  private boolean isListView = true;
13  //当前显示的安装程序
14  private List<PackageInfo> showPackageInfos = null;
```

10.3.2　ListViewAdapter 和 GridViewAdapter

接下去先介绍一下 ListViewAdapter 和 GridViewAdapter，因为在主体程序中需要用到。如下所示，是 ListViewAdapter 的代码。它继承于 BaseAdapter，因此需要重写特定的几个函数，如 getCount、getItem、getItemId 和 getView。书写该适配器的关键是实现 getView，在 getView 中设置程序的 icon、应用名称和包名。

```
01  package com.guo.apkmanager;            //声明包语句
02~10 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
11  //ListView 的适配器
12  class ListViewAdapter extends BaseAdapter {
13      //用于存放应用程序信息
14      private List<PackageInfo> packageInfos = null;
15      private LayoutInflater inflater = null;
16      private  Context context = null;
17      //构造函数，初始化变量
18      public ListViewAdapter(List<PackageInfo>  packageInfos , Context
        context){
19          this.packageInfos = packageInfos;
20          this.context = context ;
21          inflater = LayoutInflater.from(context);
22      }
23      //获得应用程序的个数
24      @Override
25      public int getCount() {
26          return packageInfos.size();
```

```
27      }
28      //获得应用程序
29      @Override
30      public Object getItem(int arg0) {
31          return packageInfos.get(arg0);
32      }
33      //获得应用程序的 ID
34      @Override
35      public long getItemId(int arg0) {
36          return arg0;
37      }
38      //设置 listView 的视图
39      @Override
40      public View getView(int position, View contentView, ViewGroup parent){
41          View view = inflater.inflate(R.layout.listviewitem, null);
42          TextView appName = (TextView) view.findViewById(R.id.lv_
            item_appname);
43          TextView packageName = (TextView) view.findViewById(R.id.lv_
            item_packagename);
44          ImageView iv = (ImageView) view.findViewById(R.id.lv_icon);
45          //设置应用程序名称
46          appName.setText(packageInfos.get(position).applicationInfo.
             loadLabel(context.getPackageManager()));
47          //设置包名
48          packageName.setText(packageInfos.get(position).packageName);
49          //设置 icon
50          iv.setImageDrawable(packageInfos.get(position).
            applicationInfo.loadIcon(context.getPackageManager()));
51          return view;
52      }
53  }
```

10.3.3　实现 getView()函数

GridViewAdapter 的代码跟 ListViewAdapter 的代码很类似，都是继承一个 BaseAdapter，只有在实现 getView()函数的时候略有不同，只需要设置程序名和 icon。

```
01  package com.guo.apkmanager;              //声明包语句
02~10 行为引入相关类，这里不再列举，请阅读光盘内容。
//
……
11  //GridView 的适配器
12  class GridViewAdapter extends BaseAdapter {
13      //用于存放应用程序信息
14      private List<PackageInfo> packageInfos = null;
15      private LayoutInflater inflater = null;
                                  //inflater 的作用是将 xml 文件转换成视图
16      private  Context context = null;
17      //构造函数，初始化变量
18      public GridViewAdapter(List<PackageInfo>  packageInfos , Context
         context){
19          this.packageInfos = packageInfos;
20          this.context = context ;
21          inflater = LayoutInflater.from(context);
22      }
23      //获得应用程序的个数
```

```
24      @Override
25      public int getCount() {
26          return packageInfos.size();
27      }
28      //获得应用程序
29      @Override
30      public Object getItem(int arg0) {
31          return packageInfos.get(arg0);
32      }
33      //获得应用程序的 ID
34      @Override
35      public long getItemId(int arg0) {
36          return arg0;
37      }
38      //设置 listView 的视图
39      @Override
40      public View getView(int position, View contentView, ViewGroup parent) {
41          View view = inflater.inflate(R.layout.gridviewitem, null);
42          TextView tv = (TextView) view.findViewById(R.id.gv_item_
            appname);
43          ImageView iv = (ImageView) view.findViewById(R.id.gv_item_
            icon);
44          //设置应用程序名称
45          tv.setText(packageInfos.get(position).applicationInfo.
            loadLabel(context.getPackageManager()));
46          //设置 icon
47          iv.setImageDrawable(packageInfos.get(position).
        applicationInfo.loadIcon(context.getPackageManager()));
48          return view;
49      }
50  }
```

10.3.4　入口函数 onCreate()

接着开始分析主体程序的入口函数 onCreate()，代码如下所示。一开始，我们打开标题栏显示进度条的功能，并设置全屏显示，如代码 05～07 行所示。然后初始化界面元素，并为按键绑定监听器。最后运行线程 thread 并设置标题栏进度条可见。

在用户程序与所有程序切换按钮的监听器中，根据标志变量的值，改变按钮的背景图片，为适配器设置相应的数据并使用 Toast 提示用户当前显示的是用户程序或者是所有程序。在视图切换按钮的监听器中，只需要根据标志位设置两个视图的显示隐藏属性，同时改变按钮的背景图片。

```
01  @Override
02  public void onCreate(Bundle savedInstanceState) {
03      super.onCreate(savedInstanceState);
04      //让 title 具有显示进度的功能
05      requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
06      //全屏显示
07      getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
08      setContentView(R.layout.main);
09      //所有程序与用户程序切换按钮
10      changeCategoryBtn = (ImageButton) findViewById(R.id.ib_change_
        category);
```

```
11      changeCategoryBtn.setOnClickListener(new OnClickListener() {
12          @Override
13          public void onClick(View v) {
14              if(isUserApp){
15                  //改变按钮背景图片
16                  changeCategoryBtn.setImageResource(R.drawable.user);
17                  //用户自己的应用程序
18                  showPackageInfos =userPackageInfos;
19                  //设置标志位
20                  isUserApp = false;
21                  Toast.makeText(MainViewActivity.this, "用户自己的程序",
                     2000).show();
22              }else{
23                  changeCategoryBtn.setImageResource(R.drawable.all);
24                  //所有应用程序
25                  showPackageInfos = packageInfos;
26                  //设置标志位
27                  isUserApp = true;
28                  Toast.makeText(MainViewActivity.this, "所有的程序",
                     2000).show();
29              }
30              gridView.setAdapter(new GridViewAdapter(showPackageInfos,
                 MainViewActivity.this));
31              listView.setAdapter(new ListViewAdapter(showPackageInfos,
                 MainViewActivity.this));
32          }
33      });
34      listView = (ListView) findViewById(R.id.lv_apps);
35      changeViewBtn = (ImageButton) findViewById(R.id.ib_change_view);
36      //列表视图和网格视图切换
37      changeViewBtn.setOnClickListener(new OnClickListener() {
38          @Override
39          public void onClick(View v) {
40              if(isListView == true){
41                  listView.setAdapter(new ListViewAdapter
                    (showPackageInfos, MainViewActivity.this));
42                  //显示列表视图
43                  listView.setVisibility(View.VISIBLE);
44                  gridView.setVisibility(View.GONE);
45                  //设置标志位
46                  isListView = false;
47                  Toast.makeText(MainViewActivity.this, "当前是列表视图",
                     2000).show();
48                  changeViewBtn.setImageResource(R.drawable.list);
49              }else {
50                  //显示网格视图
51                  listView.setVisibility(View.GONE);
52                  gridView.setVisibility(View.VISIBLE);
53                  //设置标志位
54                  isListView = true;
55                  Toast.makeText(MainViewActivity.this, "当前是网格视图",
                    2000).show();
56                  changeViewBtn.setImageResource(R.drawable.grids);
57              }
58          }
59      });
60      gridView = (GridView) findViewById(R.id.gridView);
61      gridView.setOnItemClickListener(listener);
62      listView.setOnItemClickListener(listener);
```

```
63
64      Thread thread = new Thread(this);
65      thread.start();
66      //设置标题栏进度条可见
67      setProgressBarIndeterminateVisibility(true);
68  }
```

10.3.5　线程 thread

那么线程 thread 做了什么呢？代码如下所示，运行 thread.start()将会执行下面的 run()函数。在该函数中，首先是获得系统中所有的应用程序信息并存放到数组 packageInfos 中，然后根据过滤条件获得用户的应用程序信息，存放到 userPackageInfos。接着向 handler 发送信息，如代码 37 行所示，更新系统界面。为 gridView 和 listView 分别适配数据，并隐藏标题栏的进度条。

```
01  private final int SEARCH_APP = 0 ;
02  private Handler handler = new Handler() {
03      //当消息发送过来的时候会执行下面这个方法
04      public void handleMessage(android.os.Message msg) {
05          super.handleMessage(msg);
06          if(msg.what == SEARCH_APP){
07              showPackageInfos = packageInfos;
08              gridView.setAdapter(new GridViewAdapter(showPackageInfos,
                MainViewActivity.this));
09              listView.setAdapter(new ListViewAdapter(showPackageInfos,
                MainViewActivity.this));
10              //设置标题栏进度条不可见
11              setProgressBarIndeterminateVisibility(false);
12          }
13      };
14  };
15  //这个新开辟的线程主要用来把 ListView 填充满，以避免它阻塞主线程
16  @Override
17  public void run() {
18      //获得系统中的所有包
19      packageInfos = getPackageManager().getInstalledPackages
        (PackageManager.GET_UNINSTALLED_PACKAGES | PackageManager.GET_
         ACTIVITIES);
20      //实例化用户自己安装的程序
21      userPackageInfos = new ArrayList<PackageInfo>();
22      for (PackageInfo temp : packageInfos) {
23          boolean flag = false;
24          ApplicationInfo appInfo = temp.applicationInfo;
25          if ((appInfo.flags & ApplicationInfo.FLAG_UPDATED_
             SYSTEM_APP) != 0) {
26              //更新过的系统应用程序
27              flag = true;
28          } else if ((appInfo.flags & ApplicationInfo.FLAG_SYSTEM) == 0) {
29              //用户自己的应用程序
30              flag = true;
31          }
32          if (flag) {
33              userPackageInfos.add(temp);
34          }
35      }
36      //发送一个信息给主线程，让主线程把 ProgressDialog 给取消掉
```

```
37      handler.sendEmptyMessage(SEARCH_APP);
38      //不同的操作就会有不同的参数值，该参数主要用来区分不同的操作
39      //我们可以用这个值来对用户不同的操作进行区分
40
41      try {//为了看到演示效果，加上下面这句话
42          Thread.sleep(2000);
43      } catch (InterruptedException e) {
44          e.printStackTrace();
45      }
46  }
```

10.3.6　AlertDialog

本程序的另一个核心功能将在单击监听器中实现，代码如下所示。当用户单击视图中的程序图标时，将显示一个 AlertDialog，这个 AlertDialog 中包含 3 个选项，分别是“启动程序”、“详细信息”、“卸载程序”，下面我们逐个分析这 3 个选项的功能。

第一个“启动程序”，通过 startActivity(intent)来实现，这个 intent 包含了应用程序的类名和包名；第二个功能——查看程序的详细信息，通过函数 showAppDetail 实现。该函数又新建了一个 AlertDialog，在这个 Dialog 中显示程序的名称、包名、版本号、版本名等信息；第三个功能“卸载程序”同样采用发送一个 intent 的方式来实现，但这次采用了 startActivityForResult 来实现，目的是为了防止程序删除成功后，图标仍留在界面。

当执行成功后，系统将会自动调用 onActivityResult()，在这个函数里面我们重新对存储系统应用程序和用户应用程序的数组进行了初始化，并为视图重新绑定一遍适配数据，以此来达到更新界面的目的。

```
01  OnItemClickListener listener = new OnItemClickListener() {
02      @Override
03      public void onItemClick(AdapterView<?> parent, View view, int
    position, long id) {
04          //通过 position 取出对应 apk 的 packageInfo
05          final PackageInfo packageInfo = showPackageInfos.get(position);
06          //创建一个 Dialog 来进行选择
07          AlertDialog.Builder builder = new AlertDialog.Builder
            (MainViewActivity.this);
08          builder.setTitle("选项");
09          //接收一个资源的 ID
10          builder.setItems(R.array.choice,new DialogInterface
            .OnClickListener() {
11              @Override
12              public void onClick(DialogInterface dialog, int which) {
13                  switch (which) {
14                  case 0:
15                      String packageName = packageInfo.packageName;
16                      ActivityInfo activityInfo = packageInfo.
                        activities[0];
17                      //activities 数组只有在设置了 PackageManager.GET_
                        ACTIVITIES 后才会被填充
18                      //故在获取 packageInfo 时要在后面加上一个判断条件
19                      if(activityInfo == null) {
20                          Toast.makeText(MainViewActivity.this, "没有任何
                            activity", Toast.LENGTH_SHORT).show();
21                          return;
```

```
22                      }
23                      String activityName = activityInfo.name;
24                      Intent intent = new Intent();
25                      //通过包名和类名来启动应用程序
26                      intent.setComponent(new ComponentName(packageName,
                         activityName));
27                      //启动 apk
28                      startActivity(intent);
29                      break;
30                  case 1:
31                      //显示 apk 详细信息
32                      showAppDetail(packageInfo);
33                      break;
34                  case 2:
35                      Uri packageUri = Uri.parse("package:" + packageInfo.
                        packageName);
36                      Intent deleteIntent = new Intent();
37                      deleteIntent.setAction(Intent.ACTION_DELETE);
38                      deleteIntent.setData(packageUri);
39                      //采用这句话是为了解决删除完应用后，程序图标仍然存在的 Bug。它
                           会调用 onActivityResult 方法
40                      startActivityForResult(deleteIntent, 0);
41                      break;
42                  }
43              }
44          });
45          //此处设为 null，因为默认就实现了关闭功能
46          builder.setNegativeButton("取消", null);
47          builder.create().show();
48      }
49  };
50
51  @Override
52  protected void onActivityResult(int requestCode, int resultCode, Intent
    data) {
53      super.onActivityResult(requestCode, resultCode, data);
54      //获得所有 apk
55      packageInfos = getPackageManager().getInstalledPackages
        (PackageManager.GET_UNINSTALLED_PACKAGES | PackageManager.GET_
         ACTIVITIES);
56      userPackageInfos = new ArrayList<PackageInfo>();
57      for(int i=0;i<packageInfos.size();i++) {
58
59          PackageInfo temp = packageInfos.get(i);
60          ApplicationInfo appInfo = temp.applicationInfo;
61          boolean flag = false;
62          if((appInfo.flags & ApplicationInfo.FLAG_UPDATED_SYSTEM_APP) !=
            0) {
63              flag = true;
64              //FLAG_SYSTEM 表明是系统 apk
65          } else if((appInfo.flags & ApplicationInfo.FLAG_SYSTEM) == 0) {
66              //用户 apk
67              flag = true;
68          }
69          if(flag) {
70              //添加到系统 apk 数组中
71              userPackageInfos.add(temp);
72          }
73      }
```

```
74      if(isUserApp) {
75          showPackageInfos = packageInfos;
76      } else {
77          showPackageInfos = userPackageInfos;
78      }
79      gridView.setAdapter(new GridViewAdapter(showPackageInfos,
        MainViewActivity.this));
80      listView.setAdapter(new ListViewAdapter(showPackageInfos,
        MainViewActivity.this));
81  }
82  //显示 apk 的详细信息
83  private void showAppDetail(PackageInfo packageInfo) {
84      AlertDialog.Builder builder = new AlertDialog.Builder(this);
85      builder.setTitle("详细信息");
86      StringBuffer message = new StringBuffer();
87      message.append("程序名称:" + packageInfo.applicationInfo.loadLabel
        (getPackageManager()));
88      message.append("\n 包名:" + packageInfo.packageName);   //包名
89      message.append("\n 版本号:" + packageInfo.versionCode); //版本号
90      message.append("\n 版本名:" + packageInfo.versionName); //版本名
91      builder.setMessage(message.toString());
92      builder.setIcon(packageInfo.applicationInfo.loadIcon
        (getPackageManager()));
93      builder.setPositiveButton("确定", null);     //仅仅是让 Dialog 消失
94      builder.create().show();
95  }
```

如图 10.4 所示，是查看程序信息的一个效果图。



图 10.4　查看程序信息

10.4　知 识 拓 展

在本章的最后，我们用到了 startActivityForResult()函数，下面我们结合一个小例子详细讲解一下这个函数的用法。

如果想在 Activity 中得到新打开 Activity 关闭后返回的数据，需要使用系统提供的 startActivityForResult(Intent intent, int requestCode)方法打开新的 Activity，新的 Activity 关闭后会向前面的 Activity 传回数据，为了得到传回的数据，必须在前面的 Activity 中重写 onActivityResult(int requestCode, int resultCode, Intent data)方法。

首先，我们新建一个项目 startAFR，在里面新建两个 Activity。第一个 Activity 代码如下所示，当打开的 Activity 被关闭时，平台会调用前面 Activity 的 onActivityResult()方法，把存放了返回数据的 Intent 作为参数传入，并将 Intent 中的参数取出显示到 TextView 中。具体代码如下：

```
01  package com.guo.startAFR;            //声明包语句
02~08 为引入相关类，这里不再列举，请阅读光盘内容
//
……
09  public class StartAFRActivity extends Activity {
10      TextView show;
11      @Override
12      public void onCreate(Bundle savedInstanceState) {
13          super.onCreate(savedInstanceState);
14          setContentView(R.layout.main);
15          //初始化界面元素
16          show=(TextView)findViewById(R.id.show);
17          Button btnOpen=(Button)this.findViewById(R.id.open);
18          btnOpen.setOnClickListener(new View.OnClickListener(){
19              public void onClick(View v) {
20                  //得到新打开 Activity 关闭后返回的数据
21                  //第二个参数为请求码，可以根据业务需求自己编号
22                  startActivityForResult(new Intent(StartAFRActivity.this,
                    AnotherActivity.class), 1);
23              }
24          });
25      }
26      /**
27       * 为了得到传回的数据，必须在前面的 Activity 中（指 MainActivity 类）重写
          onActivityResult 方法
28       * requestCode 请求码，即调用 startActivityForResult()传递过去的值
29       * resultCode 结果码，用于标识返回数据来自哪个新 Activity
30       */
31      @Override
32      protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
33          String result = data.getExtras().getString("result");//得到新
            Activity 关闭后返回的数据
34          show.setText(result);
35      }
```

第二个 Activity 中，在关闭 Activity 之前使用函数 setResult()将数据捆绑在 intent 中发送到第一个 Activity。代码如下：

```
01  package com.guo.startAFR;                   //声明包语句
02~06 为引入相关类，这里不再列举，请阅读光盘内容
//
……
07  public class AnotherActivity extends Activity{
08      @Override
09      protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.another);
12          Button btnClose=(Button)findViewById(R.id.close);
13          btnClose.setOnClickListener(new View.OnClickListener(){
14              public void onClick(View v) {
15                  //数据是使用 Intent 返回
16                  Intent intent = new Intent();
17                  //把返回数据存入 Intent
18                  intent.putExtra("result", "Hello,I'm back!");
19                  //设置返回数据
20                  AnotherActivity.this.setResult(RESULT_OK, intent);
21                  //关闭 Activity
22                  AnotherActivity.this.finish();
23              }
24          });
25      }
```

运行结果如图 10.5、图 10.6 和图 10.7 所示。

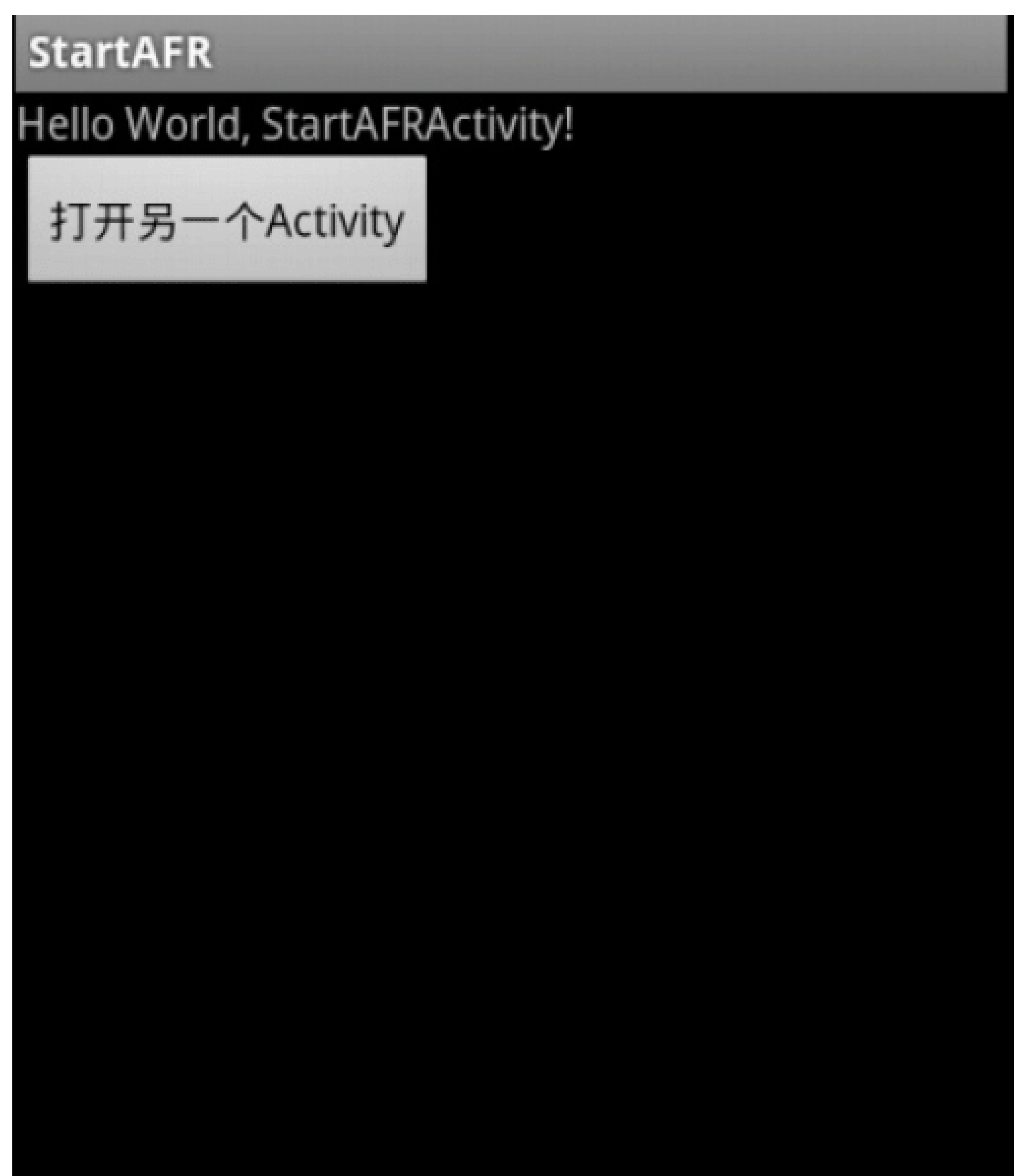


图 10.5　单击按钮之前



图 10.6　打开另一个 Activity

1．请求码的作用

使用 startActivityForResult(Intent intent, int requestCode)方法打开新的 Activity，我们需要为 startActivityForResult()方法传入一个请求码（第二个参数）。请求码的值是根据业务

需要由自已设定的，用于标识请求来源。例如，一个 Activity 有两个按钮，单击这两个按钮都会打开同一个 Activity，不管是哪个按钮打开新 Activity，当这个新 Activity 关闭后，系统都会调用前面 Activity 的 onActivityResult(int requestCode, int resultCode, Intent data)方法。onActivityResult()代码方法如果需要知道新 Activity 是由哪个按钮打开的，并且要作出相应的业务处理，这时可以这样做，代码如下所示：



图 10.7　关闭第二个 Activity 之后

```
01  @Override  public void onCreate(Bundle savedInstanceState) {
02          ....
03          button1.setOnClickListener(new View.OnClickListener(){
04              public void onClick(View v) {
05                  startActivityForResult (new Intent(MainActivity.this,
                    NewActivity.class), 1);
06              }
07          });
08          button2.setOnClickListener(new View.OnClickListener(){
09              public void onClick(View v) {
10                   startActivityForResult (new Intent(MainActivity.this,
                     NewActivity.class), 2);
11               }
12          });
13
14         @Override protected void onActivityResult(int requestCode, int
           resultCode, Intent data) {
15                switch(requestCode){
16                    case 1:
17                    //来自按钮 1 的请求，作相应业务处理
18                    case 2:
19                    //来自按钮 2 的请求，作相应业务处理
20                 }
21           }
22  }
```

2．结果码的作用

在一个 Activity 中，可能会使用 startActivityForResult()方法打开多个不同的 Activity

处理不同的业务，当这些新 Activity 关闭后，系统都会调用前面 Activity 的 onActivityResult(int requestCode, int resultCode, Intent data)方法。为了知道返回的数据来自于哪个 Activity，在 onActivityResult()方法中可以这样做，如下所示（ResultActivity 和 NewActivity 为要打开的新 Activity）：

```
01  public class ResultActivity extends Activity {
02     .....
03     ResultActivity.this.setResult(1, intent);
04     ResultActivity.this.finish();
05  }
06  public class NewActivity extends Activity {
07     ......
08      NewActivity.this.setResult(2, intent);
09      NewActivity.this.finish();
10  }
11  public class MainActivity extends Activity {
                //在该 Activity 会打开 ResultActivity 和 NewActivity
12     @Override protected void onActivityResult(int requestCode, int
       resultCode, Intent data) {
13             switch(resultCode){
14                 case 1:
15                 //ResultActivity 的返回数据
16                 case 2:
17                  //NewActivity 的返回数据
18              }
19        }
20  }
```

10.5　本 章 小 结

本章介绍了如何实现一个软件管理器，通过这个软件管理器可以集中显示系统中所有的应用程序，并可以根据用户喜好，显示成列表形式或者网格形式，也可以只显示用户应用程序。此外，通过单击窗口中的应用程序，可以查看程序的详细信息，也可以打开和卸载程序。本章最后详细介绍了 startActivityForResult 的用法。

第 3 篇　Android 网络应用实战案例

第 11 章　Android 公交查询

现代的交通越来越方便了，特别是在大城市，公交、地铁的网络已经覆盖到城市的各个角落。城市交通给我们带来便利的同时，也带来了困扰，有时候我们出门经常会纠结于乘坐哪路公交车、如何转乘等问题上。若能在手机中方便地使用公交查询的功能，将是一件让人身心愉悦的事情。本章将讲解如何开发一款基于 Android 的公交查询软件。

11.1　功 能 分 析

公交查询主要涉及以下几个方面的功能：

（1）城市选择，如图 11.1 所示。

（2）线路查询。

（3）站点查询。

（4）站-站查询，如图 11.2 所示。

（5）结果显示，如图 11.3 所示。



图 11.1　城市选择



图 11.2　站-站查询

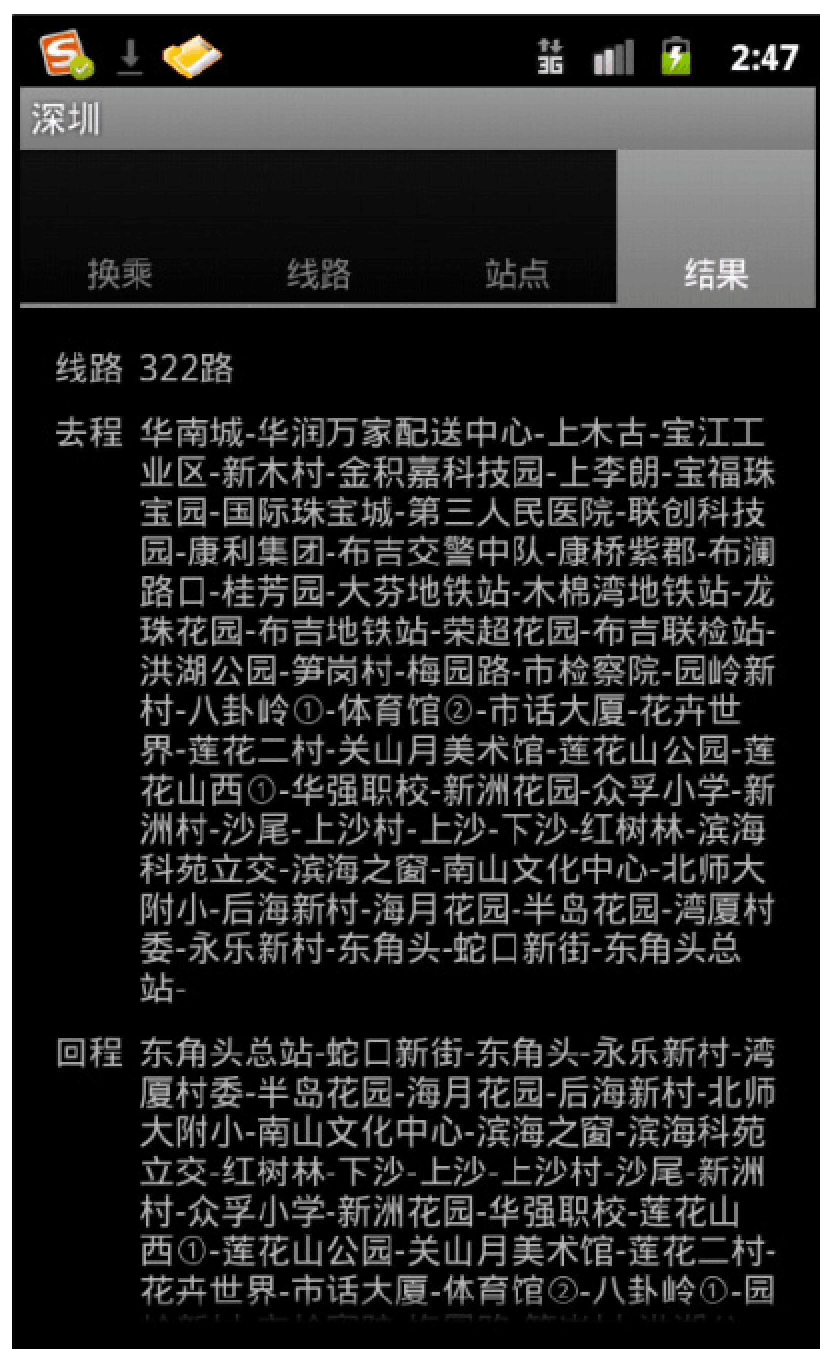


图 11.3　结果显示

公交查询很重要的一点是数据的来源，数据来源大体上可以分为两种：一是网络数据，二是离线数据。网络数据的好处是实时、准确，缺点是需要联网，网络情况差或者没有网络的时候就不能使用。离线数据的好处是不需要联网，数据也相对可靠，缺点就是如果更新不及时就会出现数据不准确的情况。

由于采用网络的方式，对网络、网站的依赖很明显，我们这里采用的是离线数据方式。下面对这几个功能进行简单说明。

城市选择：运行程序之后用户见到的第一个页面，以列表方式显示城市的数据，可以有多级子目录。单击之后进入相应的城市公交查询。

线路查询：输入公交线路，显示该线路的公交信息，需要对模糊查询作处理。例如，输入 32，这时候如果有 321、322、32、324 等线路，应该提示让用户选择。

站点查询：输入公交站点，显示经过该站点的所有线路，用户可以选择所要查询的线路。

站-站查询：提供两个输入框，用户输入起始站和终点站，系统根据算法自动算出线路。

结果显示：该页面用来显示用户的查询结果。

11.2　界 面 设 计

这个界面设计相对比较简单，一个是显示城市列表的界面，一个是用户输入信息的界面。显示城市列表的界面我们用 ListView 来实现，布局为 layout 中的文件 row.xml，代码如下：

```
01  <?xml version="1.0" encoding="UTF-8"?>
```

```
02 <LinearLayout android:orientation="horizontal"
03  android:layout width="fill parent" android:layout height="fill
    parent"
04 xmlns:android="http://schemas.android.com/apk/res/android">
05 <!-- 用于显示城市名称前的 icon -->
06   <ImageView
07       android:id="@+id/icon"
08       android:layout_width="48.0dip"
09       android:layout_height="48.0dip"
10       android:layout_marginRight="2.0dip"
11       android:layout_alignParentLeft="true" />
12 <!--用于显示城市名称 -->
13   <TextView
14       android:textSize="17.0dip"
15       android:textColor="#ffffffff"
16       android:layout gravity="center vertical"
17       android:id="@+id/label"
18       android:layout width="0.0dip"
19       android:layout height="wrap content"
20       android:text="TextView"
21       android:layout_weight="1.0" />
22 </LinearLayout>
```

这里面定义了一个 ImageView 加 TextView 组成的行，前者显示一个 icon，后者显示城市的名称。

用户输入信息界面定义在 layout 的 tab.xml，4 个界面全部写到一个文件中，通过最上面的 4 个标签进行切换。以第一个“换乘”界面为例，代码如下：

```
01 <RelativeLayout xmlns:android="http://schemas.android.com/apk/
   res/android"
02       android:id="@+id/tab1"
03       android:layout width="fill parent"
04       android:layout height="wrap content"
05       android:padding="10dip">
06    <!-- 起始站  -->
07       <TextView android:layout_width="fill_parent"
08           android:id="@+id/tab1_tv1"
09           android:layout_height="wrap_content"
10           android:text="@string/start"
11           android:layout_alignParentTop="true"
12           android:paddingBottom="5dip" />
13     <!-- 起始站输入框  -->
14       <EditText android:id="@+id/tab1 et1"
15           android:layout width="fill parent"
16           android:singleLine="true"
17           android:layout height="wrap content"
18           android:layout below="@id/tab1 tv1" />
19     <!-- 终点站  -->
20       <TextView android:id="@+id/tab1_tv2"
21           android:layout_width="fill_parent"
22           android:layout_height="wrap_content"
23           android:padding="5dip"
24           android:layout below="@id/tab1 et1"
25           android:text="@string/end" />
26     <!-- 终点站输入框  -->
27
28       <EditText android:id="@+id/tab1_et2"
29           android:layout_below="@id/tab1_tv2"
30           android:layout_width="fill_parent"
```

```
31            android:singleLine="true"
32            android:layout height="wrap content" />
33       <!--空行，用于增大间距 -->
34         <LinearLayout android:id="@+id/tab1_lay1"
35            android:layout_width="fill_parent"
36            android:layout_below="@id/tab1_et2"
37            android:layout_height="10dip"
38            xmlns:android="http://schemas.android.com/apk/res/android" />
39       <!-- 搜索按钮  -->
40         <ImageButton android:id="@+id/tab1 b1"
41            android:layout below="@id/tab1 lay1"
42            android:layout width="100dip"
43            android:layout height="50dip"
44            android:layout centerHorizontal="true"
45            android:src="@android:drawable/ic menu search" />
46   </RelativeLayout>
```

这里面主要定义了两个 EditText 用于用户输入信息，用户单击 ImageButton 之后开始搜索。

其他界面如线路查询和站点查询也都类似。结果显示页面采用 TableLayout 的布局，使数据显得更有条理性，也易于组织内容。

11.3　功 能 设 计

11.3.1　数据文件生成和校验

既然我们是查询，那就必然需要以数据为基础。为了保证数据的准确性，我们每次启动程序的时候需要去做一些检查和校验，判断数据是否存在和完整。我们一开始会将数据存放在一个称为 data.zip 的压缩包中，这个压缩包会放在 assets 目录中。之后程序启动的时候我们需要先去检查目标目录是否有数据文件，如果没有，我们就需要将压缩包的内容解压到指定目录。否则我们将会对目标文件进行文件校验，确认文件是没有被修改过的，与 data.zip 的文件一致。这样，我们的数据文件就准备完成了。

（1）运行程序首先进入 MainActivity.java：

```
1  /** Called when the activity is first created. */
2   @Override
3   public void onCreate(Bundle icicle) {
4       super.onCreate(icicle);
5       initFileNameMap();      //初始化一些全局常量
6       unzip();               //数据文件生成和校验
7   }
```

（2）因为 zip lib 不支持中文，因此 zip 里面的文件我们以英文名称命名，但是界面中又要显示中文，因此我们要创建一个哈希映射表将英文映射成中文，如下所示：

```
01  //因为 zip lib 不支持中文，因此我们要定义一个 hashtable 来对中英文进行一个映射
02  final  Hashtable<String, String> mFileName = new Hashtable<String,
    String>();
03  private void initFileNameMap() {
04     mFileName.put("shanghai", this.getResources().getString(R.string.
```

```
    shanghai));                //上海
05      mFileName.put("beijing", this.getResources().getString(R.string.
    beijing));                 //北京
06      mFileName.put("guangzhou", this.getResources().getString(R.string.
    guangzhou));               //广州
07      mFileName.put("shenzhen", this.getResources().getString(R.
    string.shenzhen));         //深圳
08      mFileName.put("chengdu", this.getResources().getString(R.string.
    chengdu));                 //成都
09      mFileName.put("fuzhou", this.getResources().getString(R.
     string.fuzhou));          //福州
10      mFileName.put("hefei", this.getResources().getString
    (R.string.hefei));         //合肥
11      mFileName.put("wuhan", this.getResources().getString
    (R.string.wuhan));         //武汉
12      mFileName.put("zhixiashi", this.getResources().getString
    (R.string.zhixiashi));    //直辖市
13  }
```

（3）之后进入 unzip()函数的主体，主要执行了 DataDownloader 的构造函数里面的内容：

```
01  private void unzip() {
02      Log.v(Globals.TAG, "start unzip file");
03      class CallBack implements Runnable {
04          public MainActivity mParent;//初始化一个主界面类
05
06          public void run() {
07              if (mParent.downloader == null)
08                //初始化一个下载类
09                  mParent.downloader = new DataDownloader(mParent,
                    mDialog);
10          }
11      }
12      CallBack cb = new CallBack();
13      cb.mParent = this;
14      mDialog = CreateDialog();
15      mDialog.show();
16      this.runOnUiThread(cb); //注意此处需要将线程运行在主线程中
17  }
```

（4）DataDownLoader 是我们自己构造的一个类，这个类继承于 Thread，我们在这个类的构造函数中运行这个线程的核心函数 run()：

```
01  //核心函数
02  @Override
03  public void run()
04  {
05      //检查目标目录的文件是否完整和正确，传入压缩文件名和要检查的标识文件名
06      if( ! DownloadDataFile(Globals.DataDownloadUrl, "DownloadFinished.
        flag") )
07      {
08          DownloadFailed = true;
09          return;
10      }
11      //如果运行到了这里，说明数据是正确的
12      DownloadComplete = true;
13      //初始化
14      initParent();
```

```
15  }
```

（5）在这个函数里面我们首先会去读取全局常量 Globals.DataDownloadUrl，在这个常量里面我们存储了数据的文件名。接下来进入到这个过程中最重要、最核心的步骤：解压和校验，也就是 DownloadDataFile 函数里面的内容。这个函数的开始，我们先去检查一个标志文件，这个文件是我们前一次解压完数据后生成的。

```
01  //初始化资源实例
02  Resources res = Parent.getResources();
03  //检查目标文件是否包含指定的数据
04  String path = getOutFilePath(DownloadFlagFileName);
05  InputStream checkFile = null;
06  try {
07      checkFile = new FileInputStream( path );
08  } catch( FileNotFoundException e ) {
09  } catch( SecurityException e ) { };
10  if( checkFile != null )
11  {
12      try {
13          //构造一个比标准数据稍大的 buffer，用于存储文件数据
14          byte b[] = new byte[ Globals.DataDownloadUrl.getBytes("UTF-8")
            .length + 1 ];
15          int readed = checkFile.read(b);
16          String compare = new String( b, 0, readed, "UTF-8" );
            //DataDownloadUrl=data.zip
17          boolean matched = false;
18          //若 compare=data.zip
19          if( compare.compareTo(DataDownloadUrl) == 0 )
20              matched = true;
21          //如果不匹配，抛出异常，直接跳转到 1 所在的位置
22          if( ! matched )
23              throw new IOException();
24          Status.setText( res.getString(R.string.download unneeded) );
25          return true;
26      } catch ( IOException e ) {};
27  }
```

（6）在这个文件里面，我们会写上压缩文件的文件名，我们每次开始的时候会检查里面的文件名是否正确。若正确，表明上次解压完成，直接进入下一步。否则，则说明上次解压过程没有完成，需要进一步检查目标目录的内容。世事难料，我们总要尽量多地考虑一些意外状况，这样我们的程序才会更健壮。

```
01  try {
02      //打开 assets 目录下的文件
03      stream = new CountingInputStream(Parent.getAssets().open(url),
         8192);
04  while( stream.skip(65536) > 0 ) { };
05  //取得文件的大小信息
06      totalLen = stream.getBytesRead();
07  stream.close();
08  //再次打开文件，将类的变量重置
09      stream = new CountingInputStream(Parent.getAssets().open(url),
        8192);
10  } catch( IOException e ) {
11  System.out.println("Unpacking from assets '" + url + "' - error: " +
    e.toString());
12  //显示提示信息
```

```
13     Status.setText( res.getString(R.string.error dl from, url) );
14     return false;
15 }
```

（7）在这里我们通过一个类 CountingInputStream 读取我们的压缩文件，并记录文件的总大小保存到 totalLen 这个变量中。这个类是我们自己创建的，继承于 BufferedInputStream，主要用于读取文件，每次读取文件都会记录当前读取到的文件的位置。

```
01 class CountingInputStream extends BufferedInputStream {
02     private long bytesReadMark = 0;              //已读字节数标识
03     private long bytesRead = 0;                  //已读字节数
04
05     public CountingInputStream(InputStream in, int size) {
06         super(in, size);
07     }
08     public CountingInputStream(InputStream in) {
09         super(in);
10     }
11     public long getBytesRead() {
12         return bytesRead;
13 }
14 //每次读一个字节
15     public synchronized int read() throws IOException {
16         int read = super.read();
17         if (read >= 0) {
18             bytesRead++;                  //字节计数加 1
19         }
20         return read;
21 }
22 //使用 synchronized 关键字保证函数每次只能由一个线程同时访问
23     public synchronized int read(byte[] b, int off, int len) throws
       IOException {
24         int read = super.read(b, off, len);
25         if (read >= 0) {
26             bytesRead += read;            //字节计数增加 read
27         }
28         return read;
29     }
30     public synchronized long skip(long n) throws IOException {
31         long skipped = super.skip(n);
32         if (skipped >= 0) {
33             bytesRead += skipped;     //字节计数增加 skipped
34         }
35         return skipped;
36     }
37     public synchronized void mark(int readlimit) {
38         super.mark(readlimit);
39         bytesReadMark = bytesRead;   //保存当前的字节计数到 bytesReadMark
40     }
41     public synchronized void reset() throws IOException {
42         super.reset();
43         bytesRead = bytesReadMark;
44     }
45 }
```

（8）接下来我们要解压并复制压缩文件里面的内容，我们使用 ZipInputStream 来读取压缩文件。因为我们目标文件的目录结构跟压缩文件的目录结构一模一样，因此我们通过读取压缩文件里面的文件名称，就能间接得到将要生成的目标文件的路径。接下去需要做

的就是文件校验：

```
01  try {
02      //使用 CRC32 进行校验
03      CheckedInputStream check = new CheckedInputStream( new
        FileInputStream(path), new CRC32() );
04      while( check.read(buf, 0, buf.length) > 0 ) {};
05  check.close();
06  //文件检验失败
07      if( check.getChecksum().getValue() != entry.getCrc() )
08      {
09          File ff = new File(path);
10          ff.delete();                //删除文件
11          throw new Exception();      //go to catch
12      }
13      System.out.println("File '" + path + "' exists and passed CRC check
- not overwriting it");
14      continue;                       //若校验成功，则进入下一个文件的校验
15  } catch( Exception e )
16  {
17  }
```

（9）通过比较文件的 SRC 校验值，来判断两个文件是否完全相同，不相同则删除当前文件，复制新的文件替代。否则继续校验下一个文件。

```
01  //读取文件，并写入目标文件
02  int len = zip.read(buf);
03  while (len >= 0)
04  {
05      if(len > 0)
06          out.write(buf, 0, len);       //写入文件
07      len = zip.read(buf);
08
09      percent = 0.0f;                   //百分比初始化为 0
10      if( totalLen > 0 )
11          percent = stream.getBytesRead() * 100.0f / totalLen;
                                          //动态显示完成的百分比
12      Status.setText( res.getString(R.string.dl progress, percent,
         path) );
13  }
14  out.flush();                          //清除缓存
15  out.close();
16  out = null;                           //释放资源
17  } catch( java.io.IOException e ) {
18  Status.setText( res.getString(R.string.error write, path) );
19  System.out.println("Saving file '" + path + "' - error writing or
    downloading: " + e.toString());
20  return false;
21  }
```

当然，我们在写完文件的时候仍要对该文件进行一次校验，若出现错误直接返回 false。

当全部写完并且校验成功之后，就会生成我们一开始提到的校验文件，并往里面写入一个字符串，表明这个操作成功完成了。

经过这么一连串操作，我们的数据就比较可靠了，能够为后面的查询做好铺垫。前面我们提到的还有一种数据获取方式——网络，其实原理是差不多的，只是把从压缩包解压复制文件换成从网络中下载保存文件。

11.3.2　显示城市列表

（1）数据准备完成之后，我们需要将城市的信息呈现给界面，供用户选择操作。

```
01  //若前面的数据都正确，则执行界面的初始化，将数据文件以列表的形式呈现出来
02  private void initParent()
03  {
04      class Callback implements Runnable
05      {
06          public MainActivity Parent;
07          public void run()
08          {
09              Parent.getFileList();     //获得目标目录的文件列表
10              Log.e("guojs","initParent!");
11          }
12      }
13      Callback cb = new Callback();
14      synchronized(this) {
15          cb.Parent = Parent;       //传入主界面类的实例
16          if(Parent != null)
17              Parent.runOnUiThread(cb);
                                      //因为需要更新界面，因此需要运行在 UI 线程
18      }
19  }
```

（2）因为这个过程可能会比较耗时，为了防止主界面阻塞，出现 ANR 错误，我们使用一个线程调用获取文件列表的函数 getFileList。getFileList 会去调用 browseTo 进行进一步的操作，如果传入的是目录，则显示目录的内容；如果为文件，则为文件添加单击弹出对话框的功能。

```
01  public void getFileList() {
02      mDialog.dismiss();                              //消除进程对话框
03      browseTo(new File(Globals.DataDir),0);      //显示文件列表
04  }
05      //显示文件列表
06  private void browseTo(final File aDirectory, final long id) {
07      //如果传入的参数是个目录，则显示目录下的所有内容
08      if (aDirectory.isDirectory()) {
09          this.currentDirectory = aDirectory;
10          fill(aDirectory.listFiles());
11      } else {
12          //如果传入的是文件，则为文件，添加按钮
13          DialogInterface.OnClickListener okButtonListener = new
            DialogInterface.OnClickListener() {
14             @Override
15              public void onClick(DialogInterface arg0, int arg1) {
16                  try {
17                      try {           //如果用户选择“确定”按钮，则进入查询界面
18                          Intent in = new Intent(MainActivity.this,
19                                  Traffic.class);
20                          //将数据文件的路径作为参数传递过去
21                          in.putExtra(Globals.FILENAME, aDirectory.
                          getPath());
22                          //将数据文件对应的中文城市名传递过去
23                          in.putExtra(Globals.Title, directoryEntries.get
```

```
                            ((int)id).mChineseName);
24                            //打开查询界面的 Activity
25                            MainActivity.this.startActivity(in);
26                        } catch (Exception e) {
27                            Context context = getApplicationContext();
28                            //如果出错，则以 Toast 方式提示用户
29                            CharSequence text = MainActivity.this
30                                    .getResources().getString(
31                                            R.string.diag err);
32                            int duration = Toast.LENGTH SHORT;
33
34                            Toast toast = Toast.makeText(context, text,
35                                    duration);
36                            toast.show();
37                        }
38                        ;
39                    } catch (Exception e) {
40                        e.printStackTrace();
41                    }
42                }
43            };
44            //添加“取消”按钮的功能
45            DialogInterface.OnClickListener cancelButtonListener = new
              DialogInterface.OnClickListener() {
46                @Override
47                //如果选择取消，则将 dialog 隐藏，不作其他操作
48                public void onClick(DialogInterface dialog, int which) {
49                    dialog.dismiss();
50                }
51            };
52            //创建一个 AlertDialog，当用户单击城市的时候会弹出提示，供用户进一步选择
53            AlertDialog ad = new AlertDialog.Builder(this).setMessage(
54                    R.string.diag msg).setPositiveButton(android.R.
                    string.ok,
55                    okButtonListener).setNegativeButton(
56                    android.R.string.cancel, cancelButtonListener)
                    .create();
57            ad.show();//显示对话框
58        }
59    }
```

（3）接下来我们需要考虑的是按照什么顺序显示这些数据，这些数据可能是目录也可能是文件。这里我们用首个汉字拼音的字典顺序，也就是 ABCDE…这样的顺序显示。既然是要按一定的顺序显示，那就必然涉及到一个排序的过程。为此我们用一个变量 directoryEntries，来存储指定目录下的文件信息：

```
private ArrayList<RowModel> directoryEntries = new ArrayList<RowModel>();
```

（4）这里出现了一个新的类型 ArrayList<RowModel>，这是我们自己定义的，用于存放文件名相关信息。代码如下：

```
01  class RowModel {
02      int mRowtype;               //0：文件 1：目录
03      String mLabel;              //英文名称
04      String mChineseName;        //中文名称
05
06      RowModel(int type, String label) {
07          mRowtype = type;
```

```
08          mChineseName = mLabel = label;
09          String temp = mFileName.get(label);
10          //若有中文名称则将中文名称存储到 mChineseName
11          if (temp != null) {
12              mChineseName = temp;
13          }
14      }
15      //返回中文名称
16      public String toString() {
17          return mChineseName;
18      }
19  }
```

（5）这样我们每次就将读取到的指定目录的内容存放进这个数组中。需要注意的一点是，每次更新目录的信息时需要清除数组中的元素。当需要的元素全部存到该数组时，需要新建一个中文比较器对数组的内容排一下序。然后将排完序的内容显示到列表中。

```
01  private void fill(File[] files) {
02      //清除数组中的所有元素
03      this.directoryEntries.clear();
04      int type = 0;
05      for (File file : files) {
06          //不是数据文件也不是目录的跳过
07          if (!file.getName().endsWith(".txt") && !file.isDirectory())
08              continue;
09          final String name;
10          //如果是文件则将文件名去掉扩展名之后保存在数组中
11          if (!file.isDirectory()) {
12              name = file.getName().substring(0,
13                      file.getName().lastIndexOf('.'));
14              type = 0;
15          } else {
16              //如果是目录则直接保存目录名
17              type = 1;
18              name = file.getName();
19          }
20          this.directoryEntries.add(new RowModel(type, name));
21      }
22      //新建一个中文字符比较器
23      Comparator<RowModel> cmp = new ChinsesCharComp();
24      //对数组文件进行排序
25      Collections.sort(directoryEntries, cmp);
26      //新建一个 IconAdapter，包含文件列表
27      IconAdapter directoryList = new IconAdapter(directoryEntries);
28      //设置 ListAdapter
29      this.setListAdapter(directoryList);
30  }
```

（6）到了这里还有几项功能没有完成：一个是为每一行的内容绑定一个监听器，一个是构建一个 ListView 的适配器，还有就是进入子目录退回的功能。

```
01  //为选项绑定监听器
02  @Override
03  protected void onListItemClick(ListView l, View v, int position, long
    id) {
04
05      File clickedFile = null;
06      //如果是一个目录，就进一步显示目录的内容
```

```
07      if(this.directoryEntries.get(position).mRowtype == 1) {
08
09          Log.v(Globals.TAG, "is a directory");
10          //通过 directoryEntries 数组的内容获取文件路径信息
11          clickedFile = new File(this.currentDirectory.getAbsolutePath()
12                  + File.separator + this.directoryEntries.get(position)
                    .mLabel);
13          this.browseTo(clickedFile, id);
14          return;
15      }
16      //如果是个文件，要先补全文件的完整路径，再打开相应的数据文件
17      clickedFile = new File(this.currentDirectory.getAbsolutePath()
18              + File.separator + this.directoryEntries.get(position)
                .mLabel + ".txt");
19      //再次确认文件是完整的
20      try {
21          if (clickedFile != null && clickedFile.isFile())
22              this.browseTo(clickedFile, id);
23
24      } catch (Exception e) {
25          //don't throw
26      }
27
28  }
29  //构建一个 ListView 的适配器
30  class IconAdapter extends ArrayAdapter<RowModel> {
31      //适配器初始化函数，设定数据来源和界面文件
32      IconAdapter(List<RowModel>  items) {
33          super(MainActivity.this, R.layout.row, _items);
34      }
35      //将数据添加到每一行，并设置图标
36      public View getView(int position, View convertView, ViewGroup parent) {
37          View row = convertView;
38
39          if (row == null) {
40              LayoutInflater inflater = getLayoutInflater();
41              row = inflater.inflate(R.layout.row, parent, false);
42          }
43          //设置文本
44          TextView tv= (TextView)row.findViewById(R.id.label);
45          tv.setText(directoryEntries.get(position).mChineseName);
46          ImageView iv = (ImageView)row.findViewById(R.id.icon);
47          //设置 icon
48          iv.setImageResource(R.drawable.icon);
49          return row;
50      }
51
52  }
53  class ChinsesCharComp implements Comparator<RowModel> {
54      //构建一个比较方法，传入需要比较的两个值
55      public int compare(RowModel o1, RowModel o2) {
56          String c1 = (String) o1.mChineseName;
57          String c2 = (String) o2.mChineseName;
58          //初始化一个中文字符集合
59          Collator myCollator = Collator.getInstance(java.util.Locale
            .CHINA);
60          //根据首个汉字的字典顺序排序
61          if (myCollator.compare(c1, c2) < 0)
62              return -1;
```

```
63          else if (myCollator.compare(c1, c2) > 0)
64              return 1;
65          else
66              return 0;
67      }
68  }
69  //按键时响应
70  @Override
71    public boolean onKeyDown(int keyCode, KeyEvent event) {
72      boolean result = true;
73      //判断如果按下的是 back 键
74      if (keyCode == KeyEvent.KEYCODE_BACK) {
75          //如果当前目录是 mybus 则退出程序
76          if(currentDirectory.getName().equals("mybus")) {
77              finish();
78          } else {
79              //否则显示上级目录的内容
80              browseTo(currentDirectory.getParentFile(),0);
81          }
82      }
83        return result;
84    }
```

这样第一个界面的功能就完全实现了。接下来我们将分析具体的公交查询的相关操作。

11.3.3　公交查询

单击城市界面中任意一个城市的名称，在弹出的对话框中选择“确认”，就会进入公交查询的主界面。

（1）首先，当然是要执行 onCreate 函数进行界面初始化操作。由于我们采用的是标签界面，因此首先需要调用 getTabHost()获得当前的 tabHost，如下面代码 08 行所示。然后我们把布局文件 R.layout.tab 通过 inflater 的方式导入到当前界面，接着初始化界面元素并为相应的按钮绑定监听器。为了使用户查询的时候更加顺畅，我们需要对数据做一个预处理，这个时间比较长，因此我们显示一个 Diaglog 提示“导入数据…”，当数据处理完毕之后会关掉这个 Dialog。

```
01  @Override
02  public void onCreate(Bundle savedInstanceState) {
03      super.onCreate(savedInstanceState);
04      //取得从上一级传递过来的参数
05      mFile = this.getIntent().getStringExtra(Globals.FILENAME);
                                              //获得数据文件名
06      setTitle(getIntent().getStringExtra(Globals.Title));
                                              //设置当前的标题为城市名称
07
08      tabHost = getTabHost();               //取得当前的 tabHost，用于管理标签
09
10      LayoutInflater.from(this).inflate(R.layout.tab,
11              tabHost.getTabContentView(), true);
                                              //将布局文件 tab 的内容扩展到当前界面中
12      //添加"换乘"标签
13      tabHost.addTab(tabHost.newTabSpec("tab1").setIndicator(
```

```
14              this.getString(R.string.interchage)).setContent(R.id
                .tab1));
15          //添加"线路"标签
16          tabHost.addTab(tabHost.newTabSpec("tab2").setIndicator(
17                  this.getString(R.string.line)).setContent(R.id.tab2));
18          //添加"站点"标签
19          tabHost.addTab(tabHost.newTabSpec("tab3").setIndicator(
20                  this.getString(R.string.station)).setContent(R.id.tab3));
21          tab4 = tabHost.newTabSpec("tab4").setIndicator(
22                  this.getString(R.string.result)).setContent(R.id.tab4);
23          //添加结果标签
24          tabHost.addTab(tab4);
25          //取得"换乘"标签的按钮，并绑定按键监听器
26          interSearchButton = (ImageButton) findViewById(R.id.tab1 b1);
27          interSearchButton.setOnClickListener(mGoListener);
28          //取得"线路"标签的按钮，并绑定按键监听器
29          lineSearchButton = (ImageButton) findViewById(R.id.tab2 b1);
30          lineSearchButton.setOnClickListener(mGoListener);
31          //取得"站点"标签的按钮，并绑定按键监听器
32          stationSearchButton = (ImageButton) findViewById(R.id.tab3_b1);
33          stationSearchButton.setOnClickListener(mGoListener);
34          //初始化各个标签的界面元素
35          start = (EditText) findViewById(R.id.tab1 et1);
36          end = (EditText) findViewById(R.id.tab1 et2);
37          line = (EditText) findViewById(R.id.tab2 et1);
38          station = (EditText) findViewById(R.id.tab3 et1);
39          //显示对话框，提示正在载入数据
40          mDialog = CreateDialog();
41          mDialog.show();
42          //对数据文件进行处理
43          m = new Model(this, mFile);
44          Thread t = new Thread(m);
45          t.start();
46  }
```

（2）程序运行到 t.start()，接下来我们将要去分析 Model 这个类的实现。Model 类继承于 Runnable，主要定义了两个 Hashtable 表，用于存放相关线路站点及时间信息，它们是 road 和 road_time。t.start()将直接调用 Model 的 run 函数。这个函数首先对这两个 Hashtable 进行初始化，存入相应数据。之后关掉当前界面的 dialog，表明导入数据成功。

```
01  //这个 hash 表存储信息格式如下：
02  //line name => line stations eg.
03  //1 => a-b-c-d-e-
04  final  Hashtable<String, String> road = new Hashtable<String, String>();
05  //这个 hash 表存储信息格式如下：
06  //line name => line time eg.
07  //1 =>{8:00 12:00} 起始站 {8:00 12:00} 终点站
08  final  Hashtable<String, String> road_time = new Hashtable<String,
    String>();
09  //主体函数
10  @Override
11  public void run() {
12      initRoadHash();
13      class CallBack implements Runnable {
14          public void run() {
15              parent.mDialog.dismiss();
16          }
17      }
```

```
18      CallBack cb = new CallBack();
19      parent.runOnUiThread(cb);
20  }
```

（3）接下来将会运行 initRoadHash()初始化 road 和 road_time 这两个 Hashtable，在此之前我们有必要先了解一下数据文件是怎么生成的。

数据文件可以采用很多方式生成，可以是数据库，可以是二进制文件，也可以是加入一些特殊字符进行加密的文件。我们这里采用比较简单的 txt 文件，只是为了方便做个演示，读者如果有需要可以自行根据需求生成，只要在解析的时候采取相应的规则读取就行。

我们这里采取的格式是：

线路名称<空格>站点 1-站点 2-站点 3-…-站点 n：{时间 1：-时间 2}线路类型<空格>上行起始站<空格>时间 1-时间 2<空格>下行起始站<空格>时间 1-时间 2…<其他信息>。

知道了数据的格式就好办了，接下来就根据这种格式解析数据：

```
01  //初始化特定城市的公交线路信息
02  private void initRoadHash() {
03      try {
04          String b = null;
05          String name = null;          //用于存放公交线路名，如"121 路"
06          String line = null;          //用于存放公交路线
07          String time = "";            //用于存放时间等其他信息
08          String encode = CharacterEnding.getFileEncode(new
            FileInputStream(file));      //获得文件的编码方式
09          Log.v(Globals.TAG, "encode is:" + encode);
10          if(encode.equals("GB18030"))
11              encode = "GBK";
12          Log.v(Globals.TAG, "1");
13          //将 InputStreamReader 包装成 Bufferedreader
14          BufferedReader buf = new BufferedReader(new InputStreamReader(
15                  new FileInputStream(file), encode));
16          int i=0;
17          Log.v(Globals.TAG, "2");
18          //每次读取一行
19          while ((b = buf.readLine()) != null) {
20              //System.out.println(b);
21              Log.v(Globals.TAG, ""+i);
22              b.trim();
23              //若得到的数据为空，继续读取下一条
24              if (b.length() == 0) {
25                  continue;
26              }
27              //如果字符串以"."开头，表明这是一条注释，继续读取下一条
28              if (b.startsWith(".")) {
29                  continue;
30              }
31              //若不包含空格，则继续读取下一条
32              final int spaceIndex = b.indexOf(' ');
33              if (spaceIndex == -1)
34                  continue;
35
36              //线路名为开始到第一个空格处之间的字符串
37              name = b.substring(0, spaceIndex);
38              if (name.trim().length() == 0)
39                  continue;
```

```
40              //↑ 换成 a，↓ 换乘 b，用于表示公交线路的上行、下行
41              name = name.replace(Character.toChars(8593)[0], 'a');
42              name = name.replace(Character.toChars(8595)[0], 'b');
43
44              //如果没有"："或者"："的位置不对，则继续读取下一条
45              final int colonIndex = b.indexOf(':');
46             //Log.v(Globals.TAG, "line" + colonIndex);
47              if (colonIndex == -1 || spaceIndex > colonIndex)
48                  continue;
49              //线路为第一个空格到"："之间的字符串
50              line = b.substring(spaceIndex + 1, colonIndex);
51              if (line.trim().length() == 0)
52                  continue;
53              line = line + "-";//添加一个"-"，方便其他函数的功能实现
54
55
56              time = b.substring(colonIndex + 1, b.length());
57              //将线路站点信息放到 road
58              road.put(name, line);
59              //将线路时间信息放到 road_time 中
60              road_time.put(name, time);
61              i++;
62              //若行数超过 0xFFF 即 4096 条则退出
63              if(i > 0xfff)
64                  break;
65          }
66          //关闭文件
67          buf.close();
68          Log.v(Globals.TAG, "total count:"+i);
69          //road.put("aa","fff-gg-aa-");   //线路站点信息采用这样的形式
70      } catch (Exception e) {
71          Log.v(Globals.TAG, "exception"+e.getMessage());
72          e.printStackTrace();
73      }
74  }
```

（4）到此为止，所有准备工作都已经完成，接下去就进入具体的查询。还记得前面我们定义的 4 个标签吗？是的，只有前 3 个标签具有查询功能，第 4 个标签是用来显示结果的。因此我们对前 3 个标签的搜索按钮绑定一个监听器，用于处理相应的查询业务。

```
01  //当用户单击按键的时候执行相应功能
02  private OnClickListener mGoListener = new OnClickListener() {
03      public void onClick(View v) {
04          String s1, s2;
05          if (v == (View) interSearchButton) {
06
07              s1 = start.getText().toString(); //用于存储用户输入起始站
08              s2 = end.getText().toString();   //用于存储用户输入终点站
09              if(s1.trim().length() == 0 ||
10                      s2.trim().length() == 0)
11                  return;
12              //验证输入信息的有效性
13              if (!checkTextValid(s1) || !checkTextValid(s2)
14                      || (s1.indexOf(s2) != -1) || (s2.indexOf(s1) != -1))
{
15                  //若无效则显示提示框
16                  showDialog1(DIALOG_YES_NO_MESSAGE);
17                  return;
```

```
18                }
19                //若输入合法则进行查询业务
20                processinterSearch(s1, s2);
21
22            } else if (v == (View) lineSearchButton) {
23
24                s1 = line.getText().toString();  //用于存储输入线路
25                if(s1.trim().length() == 0)
26                    return;
27                //验证输入数据的有效性
28                if (!checkTextValid(s1)) {
29                    //如果输入无效则显示提示框
30                    showDialog1(DIALOG_YES_NO_MESSAGE);
31                    return;
32                }
33                //若输入合法则进行查询业务
34                processRoadSearch(s1);
35            } else if (v == (View) stationSearchButton) {
36                //存储用户输入的站点信息
37                s1 = station.getText().toString();
38                if(s1.trim().length() == 0)
39                    return;
40                //验证用户输入信息的有效性
41                if (!checkTextValid(s1)) {
42                    //如果无效则显示提示框
43                    showDialog1(DIALOG_YES_NO_MESSAGE);
44                    return;
45                }
46                //若输入合法则进行查询业务
47                processStationSearch(s1);
48            } else {
49                Log.e(Globals.TAG, "error");
50            }
51        }
52    };
```

（5）我们首先来分析一下 processinterSearch 这个函数，这个函数根据用户输入的起始站和终点站进行查询，输出查询结果。

我们定义一个向量数组 Vector[] vv 用于存放乘车方案：

```
1        //取得乘车方案的所有信息
2        vv = m.get_road_for_inter_station(sa, sb);
```

具体的乘车方案计算方式将在 get_road_for_inter_station(sa,sb)这个函数中实现。这个函数也是我们本程序的核心和难点，里面涉及到获取直达方案和转乘方案的算法。

为了便于大家理解，这里先给大家讲解一下这个算法的原理。如图 11.4 所示，显示的是有直达方案的情况。

假设用户需要查询 A 站点到 B 站点的乘车路线。我们先想办法获得经过 A 站点的所有路线，记为集合 C1，同时也获得经过 B 站点的所有路线，记为集合 C2。根据集合的概念，C1 与 C2 的交集就是既经过 A 站点、也经过 B 站点的路线。那么我们只要求得 C1、C2 的交集，也就得到了 A 到 B 的直达路线。

现实中，我们出去乘坐公交车，一般要么有直达方案，要么就转乘一次，转乘两次及以上的情况比较少。这里作为示例，我们暂且考虑最多转乘一次的情况，转乘两次的情况

算法比较复杂，这里就不再讲解。图 11.5 显示的是需要转乘的情况。

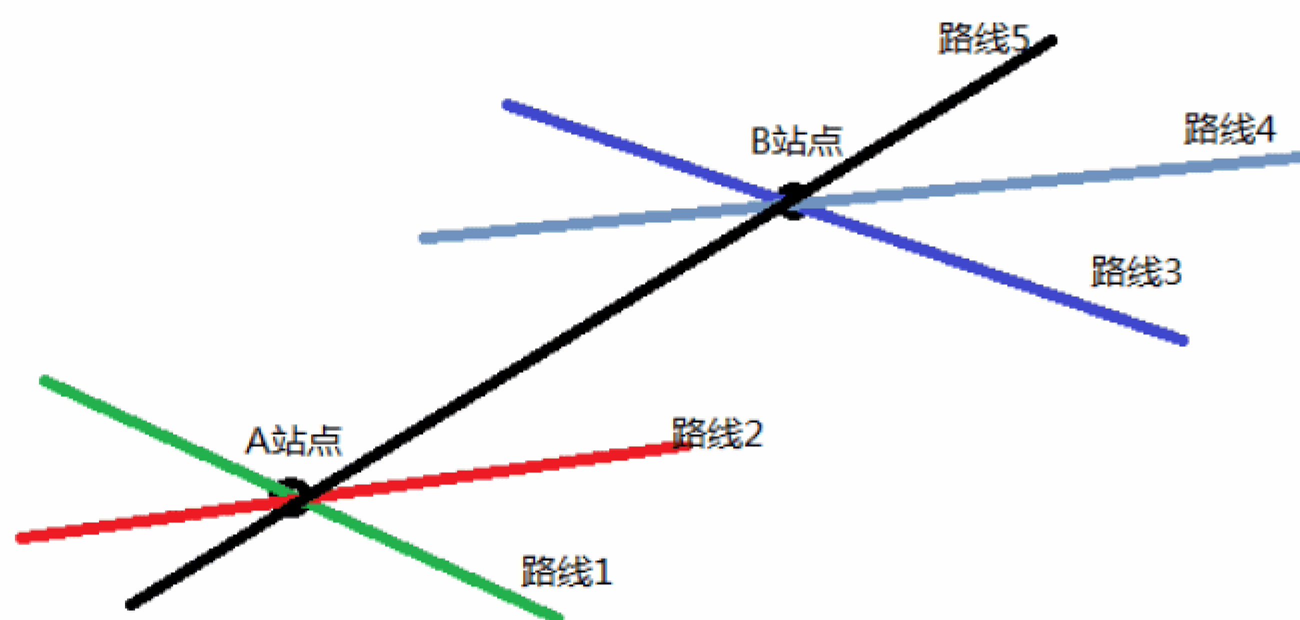


图 11.4　有直达方案的情况

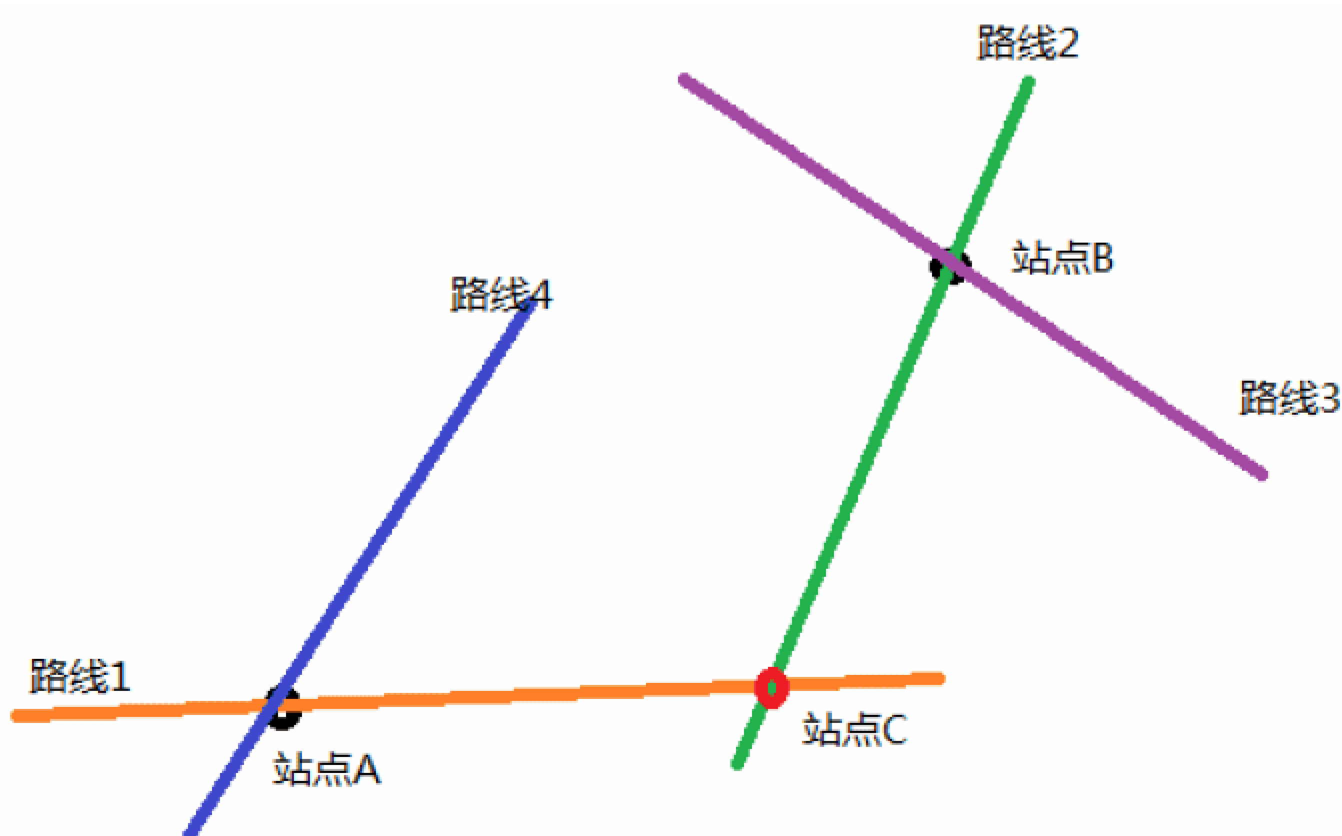


图 11.5　最少要转乘一次的情况

这种情况下 C1、C2 没有交集，也就是没有直达的方案。我们将路线抽象成一段段的线段，当然实际中应该是一条条曲线，不过这里抽象成直线的模型有助于我们更好地理解分析问题。同样，我们先得到经过 A 站点的路线集合以及经过 B 站点的路线集合，这些集合的元素可以理解成一条条的线段。我们要做的就是判断这些线段是否有交点，也就是中间站点，如果有就表明有转乘一次的方案。

经过上面的分析我们大体上知道了思路，现在就需要用程序表达出来。

```
01  //根据起点站和终点站获得乘车路线
02  public Vector[] get_road_for_inter_station(String sa, String sb) {
03      //用于存放首先要乘坐的路线
04      Vector road1 = new Vector();
05      //用于存放最后要乘坐的路线
06      Vector road2 = new Vector();
07      //用于存放中间要乘坐的路线
08      Vector road3 = new Vector();
09      Vector[] vv = new Vector[5];
10      Vector sta1 = new Vector();
11      Vector sta2 = new Vector();
12      String x = null;
13      Enumeration  direct;
14      Vector direct_vector=new Vector();
15      int index;
```

```
16      int j = 0;
17      //如果起始站与终点站相同，直接返回空
18      if (sa.trim().equals("") || sa.trim().equals("")) {
19          return null;
20      }
21      vv[0] = road1;
22      vv[1] = road2;
23      vv[2] = road3;
24      vv[3] = sta1;
25      vv[4] = sta2;
26      //取得包含起始站的所有路线信息
27      Vector tmpa = this.get_road_from_station_key(sa);
28
29      //取得包含终点站的所有路线信息
30      Vector tmpb = this.get_road_from_station_key(sb);
31      //取得所有直达路线保存到 direct_vector 中
32      direct_vector = this.get_all_the_same_string(tmpa, tmpb);
33      if(direct_vector.size() > 0)
34      {
35          //将向量 direct_vector 中的元素取出放到 direct 中
36          direct=direct_vector.elements();
37          //遍历查询 direct 中的元素
38          while(direct.hasMoreElements())
39          {
40              String a=(String)direct.nextElement();
41              road1.addElement((Object) a.toString());
42              //如果满足条件，则在 road3 中添加元素"same"表明是直达路线
43              road3.addElement((Object) "same");
44          }
45          return vv;
46      }
47      //否则表明需要转乘，seq1 存储所有包含经过起始站的公交路线
48      Enumeration seq1 = tmpa.elements();
49      //seq2 存储所有包含经过终点站的公交路线
50      Enumeration seq2;
51      //s1：第一条路 ,s2：第二条路
52      String s1, s2;
53      Vector stations1, stations2;
54      while (seq1.hasMoreElements()) {
55          //依次获取 seq1 的每个公交路线
56          s1 = (String) seq1.nextElement();
57          seq2 = tmpb.elements();
58          //检查是否有 s1 站点到终点站的直达方案，也就是转乘方案
59          while (seq2.hasMoreElements()) {
60              s2 = (String) seq2.nextElement();
61              stations1 = get_stations_for_one_road_not_care_
                direction(s1);
62              stations2 = get_stations_for_one_road_not_care_
                 direction(s2);
63              //判断这两个向量是否有交集，有则说明有转乘方案
64              x = is_there_has_a_same_string(stations1, stations2);
65              if (x != null) {
66                  j++;
67                  //最多返回 3 种转乘方案供选择
68                  if (j > 3) {
69                      return vv;
70                  }
71                  //将此时的乘车方案存储起来
72                  road1.addElement((Object) s1.toString());
```

```
                                    //road1 存储起始线路
73                  road2.addElement((Object) s2.toString());
                                    //road2 存储转乘线路
74                  road3.addElement((Object) x);       //road3 存储中转站名称
75                  sta1.addElement((Object) get_station_full_name
                    (stations1,
76                          sa.trim()));    //sta1 存储起始站点的全名，如果有的话
77                  sta2.addElement((Object) get_station_full_name
                    (stations2,
78                          sb.trim()));    //sta2 存储终点站的全名，如果有的话
79              }
80          }
81
82      }
83      //如果 road3 的元素大于一个，表明有到达方案，包括转乘和直达的
84      if (road3.size() > 0) {
85          return vv;
86      } else {
87          return null;
88      }
89  }
```

（6）如上面代码所示，我们构建了 5 个向量：road1、road2、road3、sta1、sta2，分别用来存放一个乘车方案中的起始路线、转乘路线、中转站名称、起始站实际名称、终点站实际名称。我们首先要判断是否有直达方案，如果没有再判断是否有转乘方案。函数 get_road_from_station_key(String s)用于根据站点的名称获得经过此站点的所有路线集合。

```
01  //根据站点获得经过此站点的所有路线
02  public Vector get_road_from_station_key(String s) {
03      //用于存放路线名称
04      Enumeration key_of_roads;
05      String roads;
06      int i = 0;
07      //用于存放包含指定站点的路线名称
08      Vector temp = new Vector();
09
10      String mmm = s.trim();
11      //取得线路名称的集合
12      key_of_roads = this.road.keys();
13      //遍历集合
14      while (key_of_roads.hasMoreElements()) {
15          roads = (String) key_of_roads.nextElement();
16          //如果是下行线路则跳过
17          if (roads.endsWith("b") == true)
18              continue;
19          //如果包含指定站点则存储到 temp 中
20          if (getSationsForOneRoad(roads).indexOf(mmm) != -1) {
21              if (roads.endsWith("a") == true) {
22                  //只存储路线名称，不包含上行下行信息
23                  temp.addElement((Object) roads.substring(0,
24                          roads.length() - 1));
25              } else {
26                  temp.addElement((Object) roads.toString());
27              }
28              i++;
29          }
30      }
```

```
31      return temp;
32  }
```

函数 get_road_for_inter_station(String sa, String sb)中的第 32 行 direct_vector = this.get_all_the_same_string(tmpa, tmpb)用于获得直达方案，实现代码如下：

```
01  //获得所有直达方案
02  private Vector get all the same string(Vector v1,Vector v2)
03  {
04      //用于存放直达的路线
05      Vector temp=new Vector();
06      Enumeration seq1 = v1.elements();
07      Enumeration seq2;
08      String s1, s2;
09      //遍历 seq1 的元素
10      while (seq1.hasMoreElements()) {
11          s1 = (String) seq1.nextElement();
12          //这里 seq2 需要每次都重新初始化，将当前的指向元素重置
13          seq2 = v2.elements();
14          //遍历 seq2
15          while (seq2.hasMoreElements()) {
16              s2 = (String) seq2.nextElement();
17              //若找到相同的元素，则添加该元素
18              if (s1.compareTo(s2) == 0) {
19                  temp.add((Object)s1);
20              }
21          }
22      }
23      return temp;
24  }
```

当没有直达方案的时候，我们通过算法获得转乘方案。前面已经分别得到经过 A、B 站点的路线集合，再通过遍历，判断 A 中的每一条线路是否与 B 中的每一条线路有交集。这里主要用到几个函数：get_stations_for_one_road_not_care_direction(String s1)用于根据站点名称获得与方向无关的所有线路名称，is_there_has_a_same_string(stations1，stations2)用于判断两条线路是否有交集，get_station_full_name(Vector stations，String sta)用于获取站点的全称。这 3 个函数的实现代码如下：

```
01  //不考虑上行下行，获取当前路线经过的所有站点
02  public Vector get_stations_for_one_road_not_care_direction(String s) {
03      String stations = null;
04
05      s = s.trim();
06      if (s.equals(""))
07          return null;
08      //取得当前路线的所有站点
09      stations = this.getSationsForOneRoad(s);
10      //如果没找到，尝试寻找是否有该路线上行方向的路线
11      if (stations == null) {
12          //go here mean that hashtable has two entry for this road
13          //one is end with a,another is end with b
14          stations = getSationsForOneRoad(s + "a");
15
16      }
17      int index;
18      String x;
19      //将路线依次存储到 down_load
```

```
20      Vector down_road = new Vector();
21      //通过"-"判断每个站点的名称
22      while ((index = stations.indexOf("-")) > 0) {
23          x = stations.substring(0, index);
24          down_road.addElement(x);
25          stations = stations.substring(index + 1);
26      }
27      //返回该路线经过的所有站点的信息
28      return down_road;
29
30  }
31
32  //检查两个向量中是否包含相同元素，若有则返回相同的元素，否则返回空
33  private String is_there_has_a_same_string(Vector v1, Vector v2) {
34
35      Enumeration seq1 = v1.elements();
36      Enumeration seq2;
37      String s1, s2;
38      //遍历 seq1 的元素
39      while (seq1.hasMoreElements()) {
40          s1 = (String) seq1.nextElement();
41          //这里 seq2 需要每次都重新初始化，将当前的指向元素重置
42          seq2 = v2.elements();
43          //遍历 seq2
44          while (seq2.hasMoreElements()) {
45              s2 = (String) seq2.nextElement();
46              //若找到相同的元素，则返回相同的元素
47              if (s1.compareTo(s2) == 0) {
48                  return s1;
49              }
50          }
51      }
52      return null;
53  }
54      //根据用户输入的名称获得站点的实际名称
55  private String get_station_full_name(Vector v1, String s) {
56      //获得线路的站点集合
57      Enumeration seq1 = v1.elements();
58
59      String s1, s2;
60      s = s.trim();
61      //遍历元素，判断是否有元素包含指定字符串
62      while (seq1.hasMoreElements()) {
63          s1 = (String) seq1.nextElement();
64          //返回包含指定字符串的元素
65          if (s1.indexOf(s) != -1) {
66              return s1;
67          }
68      }
69      //否则返回空
70      return "";
71  }
```

（7）现在我们所有的查询结果都存放在一个向量数组中，并赋值给之前定义的变量 vv。我们接下来要做的事就是将查询结果显示出来。因为我们的查询结果可能是多种的，需要让用户自己作一个选择，因此这里我们还要用到 dialog 这种形式来与用户交互。

```
01  //根据起始站和终点站获得乘车路线
```

```
02 private void processinterSearch(String sa, String sb) {
03     //取得乘车方案的所有信息
04     vv = m.get_road_for_inter_station(sa, sb);
05     //如果没有乘车方案，则返回回空，并提示用户没有找到
06     if (vv == null) {
07         this.showDialog1(DIALOG_YES_NO_MESSAGE);
08         return;
09     }
10     //转乘站点信息，如果有直达的，则存储的是 same
11     Enumeration seq = vv[2].elements();
12     //起始站点信息
13     Enumeration seq1 = vv[0].elements();
14     //转乘路线信息
15     Enumeration seq2 = vv[1].elements();
16     String road;
17     String road2;
18     String middleStation;
19     //初始化 road_list
20     road_list = new String[vv[0].size()];
21     //用于记录元素的个数
22     int j = 0;
23     int direct=0;
24     //取得路线的信息
25     while (seq.hasMoreElements()) {
26         middleStation = (String) seq.nextElement();
27         if (middleStation.compareTo("same") == 0) {
28             //将标志位置为 1，表明是直达
29             direct=1;
30             road = (String) seq1.nextElement();
31             Log.e("guojs",road);
32             road_list[j++]=road;
33         }else{
34             //将标志位置为 0，表明是转乘
35             direct=0;
36             road = (String) seq1.nextElement();
37             road2 = (String) seq2.nextElement();
38             road_list[j++] = road + " -> " + road2;
39         }
40     }
41     //direct =0 表明无直达方案，=1 表示有直达方案
42     if(direct == 0)
43         this.showDialog1(DIALOG_LIST1);//显示转乘方案列表
44     else
45         this.showDialog1(DIALOG_LIST_BUS);//显示直达方案列表
46 }
```

（8）我们对数据进行简单的判断和转换之后，关键的一步就是如何显示这个 dialog。核心函数就是 showDialog1(int i)，这个函数根据传入的数值不同显示不同形式的 dialog。根据需要，我们目前为止用到了 3 种形式的 dialog，一种是显示错误提示的 showDialog1(DIALOG_YES_NO_MESSAGE)，一种是显示转乘方案的 showDialog1 (DIALOG_LIST1)，还有一种是显示直达方案的 showDialog1(DIALOG_LIST_BUS)。

下面我们来看看这 3 种 dialog 分别是怎么实现的。在类的开始我们定义了 5 种类型的 dialog，如下所示：

```
1  private static final int DIALOG_YES_NO_MESSAGE = 1;//显示错误提示
2  private static final int DIALOG_LIST = 2;           //显示相似路线列表
```

```
3   private static final int DIALOG_LIST1 = 3;          //显示转乘方案列表
4   private static final int DIALOG_LIST_FOR_ROAD = 4;
                                                      //显示经过站点的路线列表
5   private static final int DIALOG_LIST_BUS = 5;   //显示直达方案列表
```

显示错误提示的 dialog 如下：

```
01  case DIALOG_YES_NO_MESSAGE:
02      return new AlertDialog.Builder(this).setIcon(
03              //设置标题：对不起；内容：没有找到相关信息
04              R.drawable.alert_dialog_icon).setTitle(R.string.sorry)
05              .setMessage(R.string.find_nothing).setPositiveButton(
06                      R.string.conform,
07                      new DialogInterface.OnClickListener() {
08                          //设置按钮不作任何操作，直接关掉对话框
09                          public void onClick(DialogInterface dialog,
10                                  int whichButton) {
11                          }
12                      }).create();
```

显示转乘方案列表的 dialog 如下：

```
01  case DIALOG_LIST1:
02      return new AlertDialog.Builder(Traffic.this).setTitle(
03              R.string.suggst_road).setItems(road_list, //设置标题：建议线路
04              new DialogInterface.OnClickListener() {
05                  public void onClick(DialogInterface dialog, int which) {
06                      //将信息显示到结果标签
07                      draw_road_table1(which);
08                      //切换当前显示的标签为结果标签
09                      tabHost.setCurrentTabByTag("tab4");
10                      dialog.cancel();
11                  }
12              }).create();
```

显示直达方案列表的 dialog 如下：

```
01  case DIALOG_LIST_BUS:
02      return new AlertDialog.Builder(Traffic.this).setTitle(
03              R.string.suggst_road).setItems(road_list,//设置标题：建议线路
04              new DialogInterface.OnClickListener() {
05                  public void onClick(DialogInterface dialog, int which) {
06                      //将信息显示到结果标签
07                      draw_road_table(road_list[which], false);
08                      //切换当前显示的标签为结果标签
09                      tabHost.setCurrentTabByTag("tab4");
10                  }
11              }).create();
```

（9）这 3 个 dialog 除了第一个不需要将结果显示到"结果”标签，其他两个都需要将数据显示到“结果”标签。根据显示形式的不同，我们需要对界面做不同的处理。比如显示转乘方案调用的是 draw_road_table1，如下所示：

```
01  //显示转乘方案的路线
02  private void draw_road_table1(int which) {
03
04      TextView line, upload, download;
05
06      table = (TableLayout) findViewById(R.id.menu);
```

```
07      if (table == null) {
08          Log.e(Globals.TAG, "tab is null");
09      }
10      //设置第一列为可收缩
11      table.setColumnShrinkable(1, true);
12      ((TextView) findViewById(R.id.tab4_h2))
13              .setText(getString(R.string.start_end)); //首末站
14      ((TextView) findViewById(R.id.tab4_h3))
15              .setText(getString(R.string.chufa1));    //先乘坐
16      ((TextView) findViewById(R.id.tab4_h4))
17              .setText(getString(R.string.daoda1));    //再乘坐
18      ((TextView) findViewById(R.id.tab4_h5)).setText("");
                                                  //设置不需要显示内容为空
19      ((TextView) findViewById(R.id.tab4_et5)).setText("");
                                                  //设置不需要显示内容为空
20      line = (TextView) findViewById(R.id.tab4_et2);
21      upload = (TextView) findViewById(R.id.tab4_et3);
22      download = (TextView) findViewById(R.id.tab4_et4);
23
24      String line0 = (String) vv[0].elementAt(which);//取得当前的起始线路
25      String line1 = (String) vv[1].elementAt(which);//取得当前的转乘线路
26      String start = (String) vv[3].elementAt(which);
                                                  //取得当前的转乘站点名称
27      String middle = (String) vv[2].elementAt(which);//取得实际起始站名称
28      String end = (String) vv[4].elementAt(which);   //取得实际终点站名称
29
30      line.setText(start + " -> " + end);
31      CharSequence styledResults = Html.fromHtml(line0 + "\n\n"
32              + getString(R.string.chufa) + ":  " + start + "\n"
33              + getString(R.string.daoda) + ":  " + middle);
34      upload.setText(styledResults);     //以 html 格式显示起始站到转乘站路线
35      styledResults = Html.fromHtml(line1 + "\n\n"
36              + getString(R.string.chufa) + ":  " + middle + "\n"
37              + getString(R.string.daoda) + ":  " + end);
38      download.setText(styledResults);  //以 html 格式显示转乘站到终点站路线
39
40  }
```

而显示直达方案路线的为 draw_road_table：

```
01  //显示路线详细信息
02  private void draw_road_table(String road, boolean append) {
03      TextView line, upload, download;
04
05      table = (TableLayout) findViewById(R.id.menu);
06      if (table == null) {
07          Log.e(Globals.TAG, "tab is null");
08      }
09      //设置第一列为可收缩
10      table.setColumnShrinkable(1, true);
11
12      ((TextView) findViewById(R.id.tab4_h2))
13              .setText(getString(R.string.line));      //线路
14      ((TextView) findViewById(R.id.tab4_h3))
15              .setText(getString(R.string.upload));    //去程
16      ((TextView) findViewById(R.id.tab4_h4))
17              .setText(getString(R.string.download));  //回程
18
```

```
19      line = (TextView) findViewById(R.id.tab4_et2);
20      upload = (TextView) findViewById(R.id.tab4_et3);
21      download = (TextView) findViewById(R.id.tab4_et4);
22
23      line.setText(road);                             //显示线路名称
24      final String[] ss = m.get_stations_for_one_road(road);
                                                        //取得线路的完整路线
25      if (ss != null) {
26          upload.setText(ss[0]);                      //显示去程
27          download.setText(ss[1]);                    //显示回程
28      }
29      //取得线路时间信息
30      final String time = m.getTimeforOneRoad(road);
31      if (time != null) {
32          ((TextView) findViewById(R.id.tab4_h5))
33                  .setText(getString(R.string.time));       //时间
34          ((TextView) findViewById(R.id.tab4_et5)).setText(time);
                                                        //显示时间信息
35      }
36  }
37
38      //根据线路名称获得完整路线
39  public String[] get_stations_for_one_road(String s) {
40      //初始化字符串数组，用于存放去程和回程
41      String[] vv = new String[2];
42      String stations = null;
43
44      s = s.trim();
45      if (s.equals(""))
46          return null;
47
48      stations = this.getSationsForOneRoad(s);
49
50      if (stations != null) {
51          //说明线路不是以 a 或者 b 结尾
52          int index;
53          String x;
54          //新建一个栈用于存放线路经过的所有站点
55          Stack down_road = new Stack();
56
57          vv[0] = stations.toString();
58          //将所有站依次添加到栈中
59          while ((index = stations.indexOf("-")) > 0) {
60              x = stations.substring(0, index);
61              down_road.addElement(x);
62              stations = stations.substring(index + 1);
63          }
64
65          String st = "";
66          //将栈的元素 pop，以得到一个以相反顺序显示站点信息的字符串
67          while (!down_road.empty()) {
68              if (st.trim().equals("")) {
69                  st = (String) down_road.pop();
70              } else {
71                  st = st + "-" + (String) down_road.pop();
72              }
73          }
74          vv[1] = st;
75          return vv;
```

```
76
77      } else {
78          //运行到这里说明线路去程和回程的路线不一致
79          //一个以 a 结尾，一个以 b 结尾
80          vv[0] = getSationsForOneRoad(s + "a");
81          if (vv[0] == null)
82              return null;
83          vv[1] = getSationsForOneRoad(s + "b");
84          if (vv[1] == null)
85              return null;
86          return vv;
87      }
88
89  }
```

（10）到这里站-站查询的讲解就结束了，我们先看看效果如何，还是分两种情况。图 11.6、图 11.7、图 11.8 显示的是有直达方案的情况。



图 11.6　输入起始站和终点站



图 11.7　显示直达方案

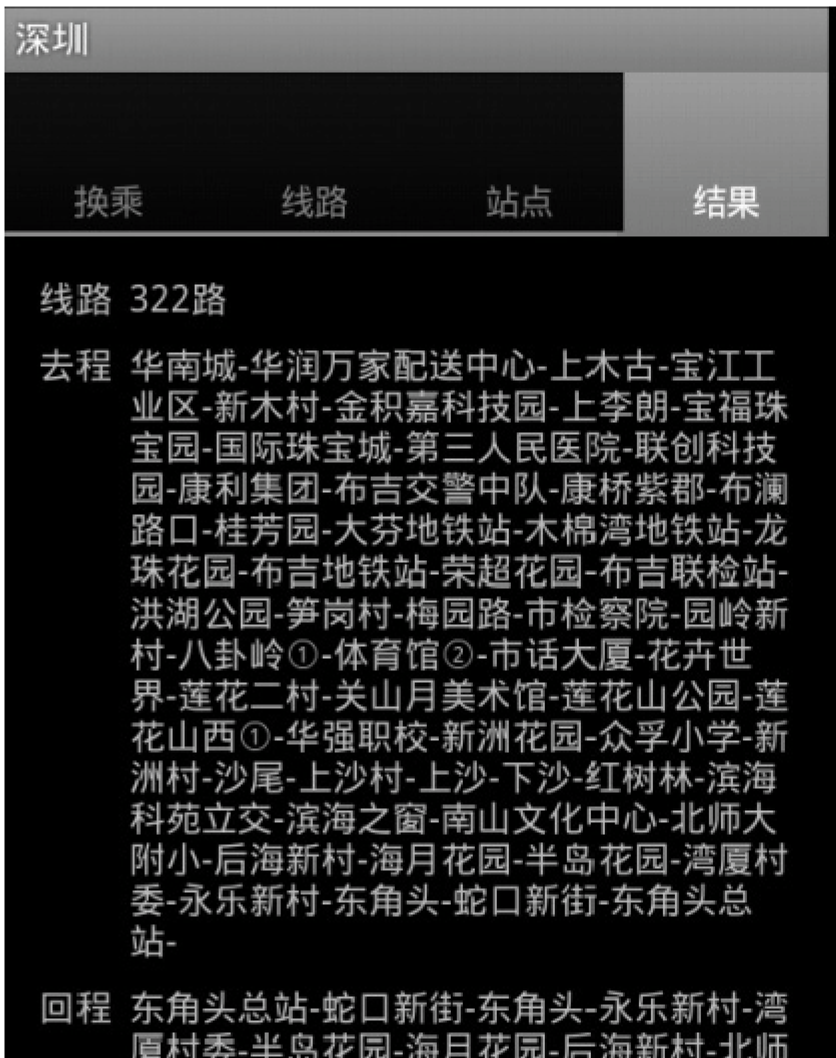


图 11.8　显示查询结果

同样我们也来看看需要转乘时的效果，如图 11.9、图 11.10、图 11.11 所示。



图 11.9　输入起始站和终点站



图 11.10　显示转乘方案



图 11.11　显示查询结果

（11）线路查询和站点查询相对比较简单，其实我们在前面换乘查询的时候也用到了这两个查询，但还是略有不同。线路查询允许模糊查询，比如输入 1，则提示你选择 1 路、112 路、123 路等等。

```
01  else if (v == (View) lineSearchButton) {
02
03  s1 = line.getText().toString();             //用于存储输入线路
04  if(s1.trim().length() == 0)
05      return;
06  //验证输入数据的有效性
07  if (!checkTextValid(s1)) {
```

```
08      //如果输入无效则显示提示框
09      showDialog1(DIALOG YES NO MESSAGE);
10      return;
11  }
12  //若输入合法则进行查询业务
13  processRoadSearch(s1);
```

（12）processRoadSearch(s1)函数中主要执行了函数 getSuggestRoads(String s)，用于获取建议路线。前面我们提到我们要做一个模糊匹配，其实就是查找以所提供字符串开头的所有线路，这里为了不重复，如果区分上行下行，我们只获取上行的路线。与换乘查询一样，我们将查询结果存储到一个向量 temp 中。

```
01  //根据路线名称获取所有建议路线
02  public Vector getSuggestRoads(String s) {
03      //用于存放路线集合
04      Enumeration key of roads;
05      String road;
06      int i = 0;
07      Vector temp = new Vector();          //用于查询存放结果
08
09      String mmm = s.trim();
10      key of roads = this.road.keys();
11      //遍历集合中的元素
12      while (key of roads.hasMoreElements()) {
13          road = (String) key_of_roads.nextElement();
14          //跳过以 b 结尾的路线
15          if (road.endsWith("b") == true)
16              continue;
17          //将开头是指定字符串的所有线路存储到元素 temp 中
18          if (road.startsWith(mmm) == true) {
19              if (road.endsWith("a") == true) {
20                  //去掉线路后面的 a，只存储线路名称
21                  temp.addElement((Object) road.substring(0,
22                          road.length() - 1));
23              } else {
24                  temp.addElement((Object) road.toString());
25              }
26              i++;
27          }
28      }
29
30      return temp;
31  }
```

（13）对查询的结果我们也稍微做了下处理，根据结果的个数，以不同的形式显示。如果是 1 个以上就显示一个列表，给用户选择；如果恰好只有 1 个，则直接显示结果到结果标签中；若没找到，提示没有相关信息。

```
01  //查询线路
02  private void processRoadSearch(String s) {
03      //用于存放建议路线
```

```
04      Vector v;
05      //取得建议路线
06      v = m.getSuggestRoads(s);
07      //如果查询结果大于 1 个，则显示一个列表供用户选择
08      if (v.size() > 1) {
09          this.list_title = this.getString(R.string.select_one_road);
10          draw_filter_list(v);
11      } else if (v.size() == 1) {
            //如果刚好是 1 个，则直接将查询结果显示到结果标签中
12          draw_road_table((String) v.elementAt(0), false);
13          this.tabHost.setCurrentTabByTag("tab4");
14      } else {
15          //如果没有结果则提示用户没找到相关信息
16          showDialog1(DIALOG_YES_NO_MESSAGE);
17      }
18  }
```

（14）我们只看一下结果大于 1 个的情况，等于 1 的情况其实就是少了一步显示列表的过程，这里不再赘述。在函数 draw_filter_list(vector v)中我们主要是显示一个新的 dialog。代码如下所示：

```
01  //将线路以列表形式显示
02  private void draw_filter_list(Vector v) {
03      Enumeration seq = v.elements();
04      int j = 0;
05      road_list = new String[v.size()];
06      //遍历查询集合的元素
07      while (seq.hasMoreElements()) {
08          road_list[j++] = (String) seq.nextElement();
09      }
10      //显示对话框
11      showDialog1(DIALOG_LIST);
12  }
```

其中，DIALOG_LIST 的形式如下：

```
01  case DIALOG_LIST:
02      return new AlertDialog.Builder(Traffic.this).setTitle( //设置标题
03              Html.fromHtml(list_title)).setItems(road_list,
                                                //以列表形式显示路线名称
04              new DialogInterface.OnClickListener() {
05                  public void onClick(DialogInterface dialog, int which) {
06                      //显示用户选定的路线的详细信息
07                      draw_road_table(road_list[which], false);
08                      //切换当前显示的标签为结果标签
09                      tabHost.setCurrentTabByTag("tab4");
10                  }
11              }).create();
```

（15）同样我们来看看运行的效果。这里我们输入 12，如图 11.12 所示，结果自动匹配出来 4 条相似的路线，如图 11.13 所示。我们随便选择一个显示，结果如图 11.14 所示。

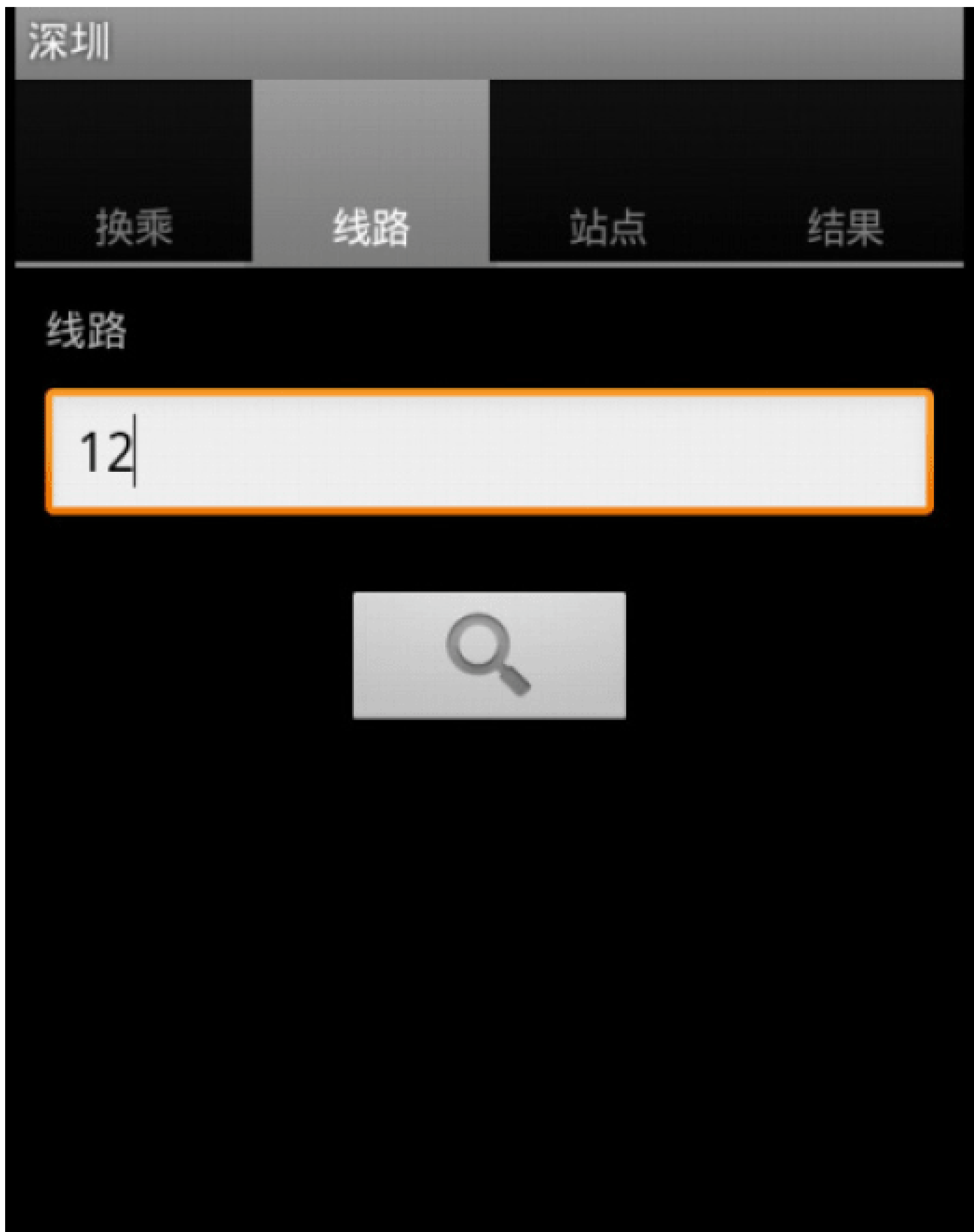


图 11.12　线路输入界面



图 11.13　显示线路列表

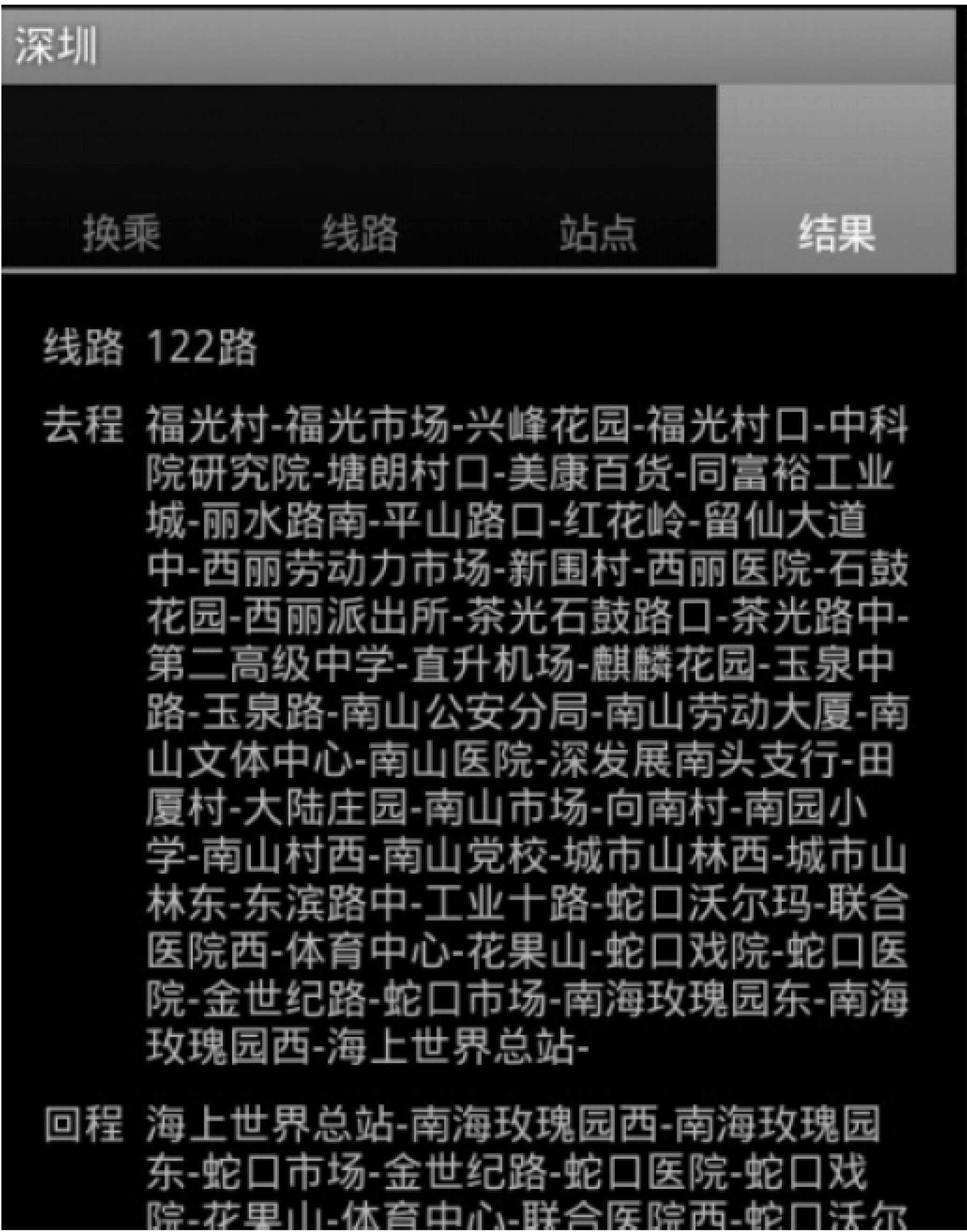


图 11.14　结果显示界面

（16）最后我们来看一下站点查询。首先还是进行简单的字符串有效性验证，接着进入核心函数 processStationSearch(s1)。

```
01  } else if (v == (View) stationSearchButton) {
02              //存储用户输入站点信息
03              s1 = station.getText().toString();
```

```
04              if(s1.trim().length() == 0)
05                  return;
06              //验证用户输入信息的有效性
07              if (!checkTextValid(s1)) {
08                  //如果无效则显示提示框
09                  showDialog1(DIALOG_YES_NO_MESSAGE);
10                  return;
11              }
12              //若输入合法则进行查询业务
13              processStationSearch(s1);
```

（17）函数 processStationSearch 中，将输入字符串首尾去除空白符之后，直接将参数输入函数 get_road_from_station_key 中，这样得到的查询结果将可能是模糊匹配的。即如果我们输入“上沙”，将会匹配所有站点中含有该字样的站点，比如“上沙”、“上沙村”。读者可以自己思考一下如何做得更好，比如先提示一个对话框，让用户从匹配到的站点选择一个站点，然后再对这个站点进行精确的查询等等。

```
01  //根据站点的名称取得经过此站点的所有线路
02  private void processStationSearch(String s) {
03      //用于存放线路名称
04      Vector mm;
05      //去掉首尾的空格
06      s = s.trim();
07      //获得经过此站点的所有路线
08      mm = m.get_road_from_station_key(s);
09      //若没找到，则提示没找到
10      if ((mm.size() == 0)) {
11          this.showDialog1(DIALOG YES NO MESSAGE);
12          return;
13      }
14      //将线路信息包进字符串变量 station_from_line 中
15      String resultsTextFormat = getString(R.string.station_from_line);
16      this.list_title = String.format(resultsTextFormat, s);
17      //显示查询结果
18      draw filter list3(mm);
19
20      return;
21  }
```

（18）函数 draw_filter_list 3 的功能是将结果显示成对话框，供用户选择，如下所示：

```
01  //用户存放站点列表
02  String[] station list;
03  //将站点列表显示到对话框中
04  private void draw_filter_list3(Vector v) {
05      //新建一个集合用于存放站点
06      Enumeration seq = v.elements();
07      int j = 0;
08      //ChoiceItem ci;
09      station list = new String[v.size()];
10      //遍历集合，将所有站点存放到 station_list 中
11      while (seq.hasMoreElements()) {
12          station_list[j++] = (String) seq.nextElement();
13      }
14      //显示结果选择对话框
15      showDialog1(DIALOG LIST FOR ROAD);
16  }
```

其中 DIALOG_LIST_FOR_ROAD 对话框的形式如下：

```
01  case DIALOG_LIST_FOR_ROAD:
02      return new AlertDialog.Builder(Traffic.this).setTitle(
03              Html.fromHtml(list_title)).setItems(station_list,//设置标题
04              new DialogInterface.OnClickListener() {
05                  public void onClick(DialogInterface dialog, int which) {
06                      //显示查询结果到"结果"标签
07                      draw_road_table(station_list[which], false);
08                      //切换到"结果"标签
09                      tabHost.setCurrentTabByTag("tab4");
10                  }
11              }).create();
```

（19）同样我们也来看一下运行效果。如图 11.15 所示，站点输入“上沙村”，单击搜索按钮将弹出结果对话框如图 11.16 所示，随便选择一路“47 路”，查看结果，如图 11.17 所示。



图 11.15　站点输入界面



图 11.16　结果对话框

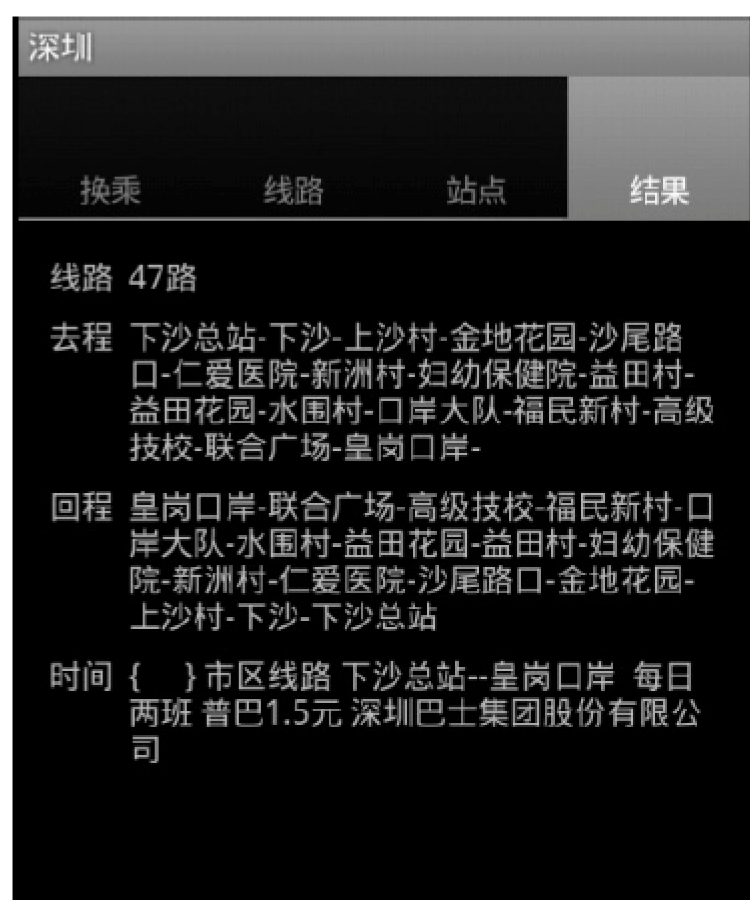


图 11.17　结果显示界面

到此为止，我们这个程序的所有功能都介绍完毕了。记得在 AndroidManifest.xml 加入读写 SD 卡的权限：

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_
STORAGE"></uses-permission>
```

11.4　知识扩展

在显示公交查询这个界面的时候我们用到了 LayoutInflater 这个类。在实际开发中 LayoutInflater 这个类还是非常有用的，它的作用类似于 findViewById()，不同点是 LayoutInflater 是用来找 layout 下的 xml 布局文件，并且实例化。而 findViewById()是找 xml 下的具体 widget 控件（如 Button、TextView 等）。

LayoutInflater 的用法有三种：

第一种方法：

```
LayoutInflater inflater = LayoutInflater.from(this);
View layout = inflater.inflate(R.layout.main, null);
```

第二种方法：

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.main, null);
```

第三种方法：

```
LayoutInflater inflater = (LayoutInflater) getSystemService(LAYOUT_
INFLATER_SERVICE);
View layout = inflater.inflate(R.layout.main, null);
```

第一种方法的本质就是调用第三种方法。

为了让大家容易理解，这里做了一个简单的 Demo，主布局 main.xml 里有一个 TextView 和一个 Button。当单击 Button，出现 Dialog，而这个 Dialog 的布局方式是我们在 layout 目录下定义的 my_dialog.xml 文件（里面上下分布，上边是 ImageView，下边是 TextView）。

my_dialog.xml 的代码如下：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
03      android:layout_width="match_parent"
04      android:layout_height="match_parent"
05      android:orientation="vertical" >
06      <!-- 用于图片 -->
07      <ImageView android:id="@+id/image"
08               android:layout_width="wrap_content"
09               android:layout_height="fill_parent"
10               android:layout_marginRight="10dp"
11               />
12      <!-- 用于显示文本 -->
13      <TextView android:id="@+id/text"
14              android:layout_width="wrap_content"
15              android:layout_height="fill_parent"
16              android:textColor="#FFF"
```

```
17              />
18  </LinearLayout>
```

主界面的代码如下：

```
01  public class LayoutInflateActivity extends Activity {
02      /** Called when the activity is first created. */
03      @Override
04      public void onCreate(Bundle savedInstanceState) {
05          super.onCreate(savedInstanceState);
06          setContentView(R.layout.main);
07          Button bt1=(Button)findViewById(R.id.btn1);
08          //为按键绑定监听器
09          bt1.setOnClickListener(new OnClickListener(){
10              @Override
11              public void onClick(View arg0) {
12                  //TODO Auto-generated method stub
13                  //显示对话框
14                  showMyDialog();
15              }
16          });
17      }
18      //显示对话框
19      public void showMyDialog()
20      {
21          AlertDialog.Builder builder;
22          AlertDialog alertDialog;     //新建一个对话框
23          Context mContext = LayoutInflateActivity.this;
                                         //取得当前程序的上下文
24
25          //下面两种方法都可以
26          //LayoutInflater inflater = getLayoutInflater();
27          LayoutInflater inflater = (LayoutInflater)
28          mContext.getSystemService(LAYOUT INFLATER SERVICE);
29          View layout = inflater.inflate(R.layout.my dialog,null);
                                             //寻找自定义的 layout
30          TextView text = (TextView) layout.findViewById(R.id.text);
31          text.setText("Hello, Welcome to Read my Book!");
                                             //设置显示的文本
32          ImageView image = (ImageView) layout.findViewById(R.id.image);
33          image.setImageResource(R.drawable.phone);    //设置显示的图片
34          builder = new AlertDialog.Builder(mContext);
35          builder.setView(layout);     //设置以 my_dialog 的格式显示对话框
36          alertDialog = builder.create();
37          alertDialog.show();          //显示对话框
38      }
39  }
```

最后执行，当单击按键的时候出现对话框，如图 11.18 所示。



图 11.18　显示自定义对话框

11.5　本章小结

现在公交查询的软件有很多，要想突出自己的特色，除了实现上述的基本功能外，还需要考虑一些创新的功能。比如可以和 GPS 及地图功能集合，显示用户当前附近的站点，以及将公交路线显示到地图上等。或者加入一些算法，计算两个站点之间的最快方案、最少转乘方案等。也可以和网络结合，分享到微博等。当然并不是功能越多越好，主要还是要加强公交查询的功能，不能喧宾夺主，什么都一箩筐塞给用户，反而让用户反感。

第 12 章　股票查询软件

股票是现代人生活中难以缺少的一个元素了，虽然亏的多赚的少，但是大家还是乐此不疲地往里头跳。作为智能手机的用户，如果没有时间时时刻刻坐在电脑前关注自己的股票，就需要借助一款好的手机股票软件，来满足出门在外对自己持有股票的关注。

12.1　功 能 分 析

要开发股票查询软件，首先我们要了解如何获得股票的数据，比较好的办法就是通过网站接口访问股票实时数据。这些数据格式一般比较简单，最重要的是统一，我们获得数据之后只需要对字符进行简单的处理和分类即可。

以新浪网站的数据为例，当我们访问 http://hq.sinajs.cn/list=sh601006 网站之后可以得到如下所示的数据：

```
var hq_str_sh601006="大秦铁
路,6.15,6.17,6.17,6.18,6.11,6.16,6.17,9572295,58802936,178049,6.16,37000
,6.15,664200,6.14,146837,6.13,247355,6.12,78511,6.17,344137,6.18,483375
,6.19,737475,6.20,368300,6.21,2012-11-22,15:03:06,00";
```

这个字符串由许多数据拼接在一起，不同含义的数据用逗号隔开，这些数据的含义依次是：

0：“大秦铁路”，股票名字；

1：“6.15”，今日开盘价；

2：“6.17”，昨日收盘价；

3：“6.17”，当前价格；

4：“6.18”，今日最高价；

5：“6.11”，今日最低价；

6：“6.16”，竞买价，即“买一”报价；

7：“6.17”，竞卖价，即“卖一”报价；

8：“9572295”，成交的股票数，由于股票交易以 100 股为基本单位，所以在使用时，通常把该值除以 100；

9：“58802936”，成交金额，单位为“元”，为了一目了然，通常以“万元”为成交金额的单位，所以通常把该值除以 10000；

10：“178049”，“买一”申请 178049 股，即 1780 手；

11：“6.16”，“买一”报价；

12：“37000”，“买二”；

13：“6.15”，“买二”报价；
14：“664200”，“买三”；
15：“6.14”，“买三”报价
16：“146837”，“买四”；
17：“6.13”，“买四”报价
18：“247355”，“买五”；
19：“6.12”，“买五”报价
20：“78511”，“卖一”申报 78511 股，即 785 手；
21：“6.17”，“卖一”报价；
(22, 23), (24, 25), (26,27), (28, 29)分别为“卖二”至“卖四的情况”；
30：“2012-11-22”，日期；
31：“15:03:06”，时间。

了解了每一个数据的含义之后，接下来我们只需要解析出这些数据，并根据我们程序的需要显示即可。图 12.1 所示为股票查询软件的主界面，每一行显示一支股票，从左到右依次是股票代码、股票名称、股票昨日收盘价以及股票当前的涨跌幅。最下面为一个文本框，如图中文本框内的提示，这个部分是用来添加股票用的。

除了主界面显示股票的大体信息外，我们设置了另一个页面用来显示股票的详细信息，包括 K 线图等，用户还可以在该界面对股票进行删除。



图 12.1　股票信息主界面

12.2　界 面 布 局

本程序的界面主要由两部分组成，一个是主页面，用于显示各个股票的简要信息，包括股票名称、股票代码和股票当前价格；另一个界面用于显示股票的详细信息，包括股票的买一、买二、卖一、卖二、当日最高价、当日最低价、日 K 线图等。

12.2.1　主界面的设置

在 res/layout 下新建 main.xml，代码如下所示，采用 RelativeLayout 布局，最上面放置一个 ListView 用于显示股票的信息，位于其下方的是一个股票添加界面，包括一个文本编辑框用于输入股票代码和一个按钮用于执行添加操作。最下方为一个 TextView，当 ListView 列表为空的时候，将显示此界面。

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <RelativeLayout
03     xmlns:android="http://schemas.android.com/apk/res/android"
04     android:id="@+id/list container"
05     android:layout width="fill parent"
06     android:layout height="fill parent">
07       <!-- 股票信息列表 -->
08       <ListView android:id="@android:id/list"
09          android:layout width="fill parent"
10          android:layout height="370dip"
11          android:layout alignParentTop="true"
12          android:choiceMode="singleChoice"
13          android:orientation="horizontal"/>
14        <!-- 位于界面最下方的界面布局 -->
15        <RelativeLayout
16          android:layout width="wrap content"
17          android:orientation="horizontal"
18          android:layout height="wrap content">
19             <!-- 股票代码添加文本框 -->
20            <EditText
21               android:id="@+id/stock_symbols"
22               android:gravity="bottom"
23               android:layout width="267dip"
24               android:layout alignParentBottom="true"
25               android:hint="@string/enter_symbols"
26               android:editable="true"
27               android:singleLine="true"
28               android:layout height="wrap content"/>
29            <!-- 股票添加按钮 -->
30            <Button
31               android:id="@+id/add symbols button"
32               android:text="@string/add"
33               android:layout_alignTop="@id/stock_symbols"
34               android:layout_alignParentRight="true"
35               android:layout width="wrap content"
36               android:layout height="wrap content" />
37         </RelativeLayout>
38          <!-- 如果列表为空时显示的界面-->
39         <TextView
40             android:id="@+id/empty"
41             android:layout width="fill parent"
42             android:layout height="fill parent"
43             android:text="@string/no stocks"/>
44  </RelativeLayout>
```

12.2.2　设置 ListView 布局

ListView 的每一行元素的布局代码如下所示，均采用 TextView 显示，从左到右分别显示股票代码、股票名称、股票当前价格、股票涨跌百分比。

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <RelativeLayout xmlns:android="http: //schemas.android.
    com/apk/res/android"
03      android:layout_width="wrap_content"
04      android:layout_height="wrap_content"
05      android:id="@+id/quote_template"
06      android:orientation="horizontal" >
07      <!-- 显示股票代码 -->
08      <TextView
09          android:id="@+id/symbol"
10          android:layout width="90sp"
11          android:layout height="wrap content"
12          android:textSize="18sp"
13          android:singleLine="true"
14          android:typeface="sans"
15          android:textStyle="bold"
16          android:paddingLeft="3sp"
17          android:paddingRight="3sp" />
18      <!-- 显示股票的名称 -->
19      <TextView
20          android:id="@+id/name"
21          android:layout width="100sp"
22          android:layout height="wrap content"
23          android:layout toRightOf="@+id/symbol"
24          android:gravity="left"
25          android:paddingLeft="3dip"
26          android:paddingRight="10dip"
27          android:singleLine="true"
28          android:textSize="18sp"
29          android:textStyle="bold"
30          android:typeface="sans" />
31      <!-- 显示股票的当前价格 -->
32      <TextView
33          android:id="@+id/current"
34          android:layout width="60sp"
35          android:layout height="wrap content"
36          android:textSize="18sp"
37          android:singleLine="true"
38          android:typeface="sans"
39          android:gravity="left"
40          android:textStyle="bold"
41          android:paddingLeft="3dip"
42          android:paddingRight="5dip"
43          android:layout toRightOf="@+id/name" />
44      <!-- 显示股票的涨跌百分比 -->
45      <TextView
46          android:id="@+id/percent"
47          android:layout_width="80sp"
48          android:layout_height="wrap_content"
49          android:textSize="18sp"
50          android:singleLine="true"
51          android:typeface="sans"
```

```
52          android:gravity="left"
53          android:textStyle="bold"
54          android:paddingLeft="3dip"
55          android:paddingRight="5dip"
56          android:layout_toRightOf="@+id/current"
57          android:layout_alignParentRight="true" />
58   </RelativeLayout>
```

12.2.3　设置界面布局

当用户单击列表中的任一行元素时，将显示所单击股票的详细信息，界面布局代码如下所示。整个布局采用 RelativeLayout 作为根元素，其中最上面显示当前股票的名称、编号、当前价格等。由于考虑到对齐的方便，因此采用 TableLayout 的布局方式将这些元素逐一放在 TableRow 中。接着下面放置一个 ImageView 用于显示股票的日 K 线图，最下面放置两个按钮，一个是删除按钮，一个是返回主界面的按钮。

```
001 <?xml version="1.0" encoding="UTF-8"?>
002 <RelativeLayout
003    android:id="@+id/quote_detail_view"
004    android:layout_width="fill_parent"
005    android:layout_height="wrap_content"
006    xmlns:android="http://schemas.android.com/apk/res/android" >
007    <!-- 采用表格布局显示股票详细信息 -->
008    <TableLayout
009       android:id="@+id/table1"
010       android:layout_width="fill_parent"
011       android:layout_height="wrap_content"
012       android:stretchColumns="0,1,2,3">
013       <TableRow
014          android:layout_width="fill_parent"
015          android:layout_height="wrap_content">
016          <!-- 显示股票当前价格标签 -->
017          <TextView
018             android:id="@+id/current_lable"
019             android:layout_width="match_parent"
020             android:layout_height="fill_parent"
021             android:gravity="left"
022             android:text="当前价格："
023             android:singleLine="true"
024             android:typeface="sans"
025             android:editable="false">
026          </TextView>
027          <!-- 显示股票当前价格 -->
028          <TextView
029             android:id="@+id/current"
030             android:layout_width="match_parent"
031             android:layout_height="fill_parent"
032             android:singleLine="true"
033             android:gravity="left"
034             android:typeface="sans"
035             android:editable="false">
036          </TextView>
037          <!-- 内容为空，用来占据位置 -->
038          <TextView
039             android:layout_width="match_parent"
040             android:layout_height="fill_parent">
```

```
041             </TextView>
042             <!-- 内容为空，用来占据位置 -->
043             <TextView
044                 android:layout_width="match_parent"
045                 android:layout_height="fill_parent">
046             </TextView>
047         </TableRow>
048          <TableRow
049             android:layout_width="fill_parent"
050             android:layout_height="wrap_content">
051             <!-- 用于显示股票编号标签 -->
052             <TextView
053                 android:id="@+id/no label"
054                 android:layout width="match parent"
055                 android:layout height="fill parent"
056                 android:gravity="left"
057                 android:text="股票编号："
058                 android:singleLine="true"
059                 android:typeface="sans"
060                 android:editable="false">
061             </TextView>
062             <!-- 用于显示股票编号 -->
063             <TextView
064                 android:id="@+id/no"
065                 android:layout width="match parent"
066                 android:layout height="fill parent"
067                 android:singleLine="true"
068                 android:gravity="left"
069                 android:typeface="sans"
070                 android:editable="false">
071             </TextView>
072             <!-- 内容为空，用来占据位置 -->
073             <TextView
074                 android:layout width="match parent"
075                 android:layout height="fill parent">
076             </TextView>
077             <!-- 内容为空，用来占据位置 -->
078             <TextView
079                 android:layout_width="match_parent"
080                 android:layout_height="fill_parent">
081             </TextView>
082         </TableRow>
083         <TableRow
084             android:layout width="match parent"
085             android:layout height="wrap content">
086             <!-- 用于显示股票今日开盘价标签 -->
087             <TextView
088                 android:id="@+id/open label"
089                 android:layout width="match parent"
090                 android:layout_height="fill_parent"
091                 android:gravity="left"
092                 android:text="今日开盘价："
093                 android:singleLine="true"
094                 android:typeface="sans"
095                 android:editable="false" >
096             </TextView>
097             <!-- 用于显示股票今日开盘价 -->
098             <TextView
099                 android:id="@+id/opening_price"
100                 android:layout_width="match_parent"
```

```
101                 android:layout_height="fill_parent"
102                 android:singleLine="true"
103                 android:gravity="left"
104                 android:typeface="sans"
105                 android:editable="false" >
106             </TextView>
107             <!-- 用于显示股票昨日收盘价标签 -->
108             <TextView
109                 android:id="@+id/volume_label"
110                 android:layout_width="match_parent"
111                 android:layout_height="fill_parent"
112                 android:gravity="left"
113                 android:text="昨日收盘价: "
114                 android:singleLine="true"
115                 android:typeface="sans"
116                 android:editable="false" >
117             </TextView>
118             <!-- 用于显示股票昨日收盘价 -->
119             <TextView
120                 android:id="@+id/closing_price"
121                 android:layout_width="match_parent"
122                 android:layout_height="fill_parent"
123                 android:singleLine="true"
124                 android:gravity="left"
125                 android:typeface="sans"
126                 android:editable="false" >
127             </TextView>
128             </TableRow>
129         <TableRow
130             android:layout_width="fill_parent"
131             android:layout_height="wrap_content">
132             <!-- 用于显示今日最低价标签 -->
133             <TextView
134                 android:id="@+id/day_low_label"
135                 android:layout_width="match_parent"
136                 android:layout_height="fill_parent"
137                 android:gravity="left"
138                 android:text="今日最低价: "
139                 android:singleLine="true"
140                 android:typeface="sans"
141                 android:editable="false">
142             </TextView>
143             <!-- 用于显示今日最低价 -->
144             <TextView
145                 android:id="@+id/day_low"
146                 android:layout_width="match_parent"
147                 android:layout_height="fill_parent"
148                 android:singleLine="true"
149                 android:gravity="left"
150                 android:typeface="sans"
151                 android:editable="false">
152             </TextView>
153             <!-- 用于显示今日最高价标签 -->
154             <TextView
155                 android:id="@+id/day_high_label"
156                 android:layout_width="match_parent"
157                 android:layout_height="fill_parent"
158                 android:gravity="left"
159                 android:text="今日最高价: "
160                 android:singleLine="true"
```

```
161                 android:typeface="sans"
162                 android:editable="false">
163             </TextView>
164             <!-- 用于显示今日最高价 -->
165             <TextView
166                 android:id="@+id/day_high"
167                 android:layout_width="match_parent"
168                 android:layout_height="fill_parent"
169                 android:singleLine="true"
170                 android:gravity="left"
171                 android:typeface="sans"
172                 android:editable="false">
173             </TextView>
174         </TableRow>
175 </TableLayout>
176     <LinearLayout
177         android:id="@+id/row4"
178         android:layout_below="@+id/table1"
179         android:layout_marginTop="5dip"
180         android:layout_width="fill_parent"
181         android:layout_height="wrap_content"
182         android:layout_marginLeft="10px"
183         android:orientation="horizontal">
184     <!-- 用于显示当日 K 线图 -->
185     <ImageView
186         android:id="@+id/chart_view"
187         android:layout_width="fill_parent"
188         android:layout_height="wrap_content">
189     </ImageView>
190     </LinearLayout>
191     <RelativeLayout
192         android:id="@+id/row5"
193         android:layout_below="@id/row4"
194         android:layout_width="fill_parent"
195         android:layout_marginTop="5dip"
196         android:layout_marginRight="15dip"
197         android:layout_marginLeft="15dip"
198         android:layout_height="65dip"
199         android:orientation="horizontal">
200         <!-- 删除当前股票 -->
201          <Button
202             android:id="@+id/delete"
203             android:layout_height="wrap_content"
204             android:layout_width="wrap_content"
205             android:layout_alignParentLeft="true"
206             android:layout_alignParentBottom="true"
207             android:text="删除"/>
208             <!-- 返回主界面 -->
209          <Button
210             android:id="@+id/close"
211             android:layout_height="wrap_content"
212             android:layout_width="wrap_content"
213             android:layout_alignParentRight="true"
214             android:layout_alignParentBottom="true"
215             android:text="返回"/>
216     </RelativeLayout>
217 </RelativeLayout>
```

12.3　功 能 实 现

12.2 节介绍了股票界面布局的设置，本节将讲解如何实现代码的编写。

12.3.1　新建一个类 StockInfo

首先我们需要新建一个类 StockInfo 用于存放股票信息，代码如下所示。这个类中主要用于存放股票的编号、名称、今日开盘价、昨日收盘价、当前价格、今日最高价、今日最低价。

```
01  package com.supermario.stocker;
02
03  public class StockInfo {
04      //股票编号
05      String no;
06      //股票名称
07      String name;
08      //今日开盘价
09      String opening price;
10      //昨日收盘价
11      String closing_price;
12      //当前价格
13      String current price;
14      //今日最高价
15      String max_price;
16      //今日最低价
17      String min_price;
18      //取得股票编号
19      public String getNo()
20      {
21          return no;
22      }
23      //设置股票编号
24      public void setNo(String no)
25      {
26          this.no=no;
27      }
28      //取得股票名称
29      public String getName()
30      {
31          return name;
32      }
33      //设置股票名称
34      public void setName(String name)
35      {
36          this.name=name;
37      }
38      //取得股票今日开盘价
39      public String getOpening_price()
40      {
41          return opening_price;
42      }
```

```
43      //设置股票今日开盘价
44      public void setOpening price(String opening price)
45      {
46          this.opening_price=opening_price;
47      }
48      //取得股票昨日收盘价
49      public String getClosing price()
50      {
51          return closing price;
52      }
53      //设置股票昨日收盘价
54      public void setClosing_price(String closing_price)
55      {
56          this.closing_price=closing_price;
57      }
58      //取得股票当前价格
59      public String getCurrent price()
60      {
61          return current price;
62      }
63      //设置股票当前价格
64      public void setCurrent_price(String current_price)
65      {
66          this.current_price=current_price;
67      }
68      //取得股票今日最高价
69      public String getMax price()
70      {
71          return max price;
72      }
73      //设置股票今日最高价
74      public void setMax_price(String max_price)
75      {
76          this.max price=max price;
77      }
78      //取得股票今日最低价
79      public String getMin price()
80      {
81          return min price;
82      }
83      //设置股票今日最低价
84      public void setMin price(String min price)
85      {
86          this.min_price=min_price;
87      }
88  }
```

12.3.2　初始化主界面

接着我们需要初始化主界面，新建 Stocker.java，代码如下所示。在初始化函数 onCreate()中首先判断存放股票的文件 symbols.txt 是否存在，如果不存在则用函数 openFileOutput()创建。接着实例化 DataHandler 类和数据适配器 QuoteAdaptor，并设置当前 ListView 的适配器为 quoteAdapter。

在 onResume()函数中执行适配器类 quoteAdapter 中的函数 startRefresh，用于更新界面。当界面不可见时调用 onStop()函数执行 stopRefresh 函数，停止更新界面。

```
01 //主界面实现类
02 public class Stocker extends ListActivity implements View.
   OnClickListener, KeyEvent.Callback {
03     //股票数据适配器
04     private QuoteAdaptor quoteAdaptor;
05     //股票代码输入框
06     private EditText symbolText;
07     //股票代码输入按钮
08     private Button addButton;
09     //返回按钮
10     private Button cancelButton;
11     //删除按钮
12     private Button deleteButton;
13     //对话框
14     private Dialog dialog = null;
15     //股票详细信息
16     private TextView currentTextView,noTextView, openTextView,
       closeTextView, dayLowTextView, dayHighTextView;
17     //日 K 线图
18     private ImageView chartView;
19     //股票数据处理类
20     DataHandler mDataHandler;
21     //当前 Activity 实例
22     Stocker mContext;
23     //当前选中的股票的序号
24     int currentSelectedIndex;
25     //初始化界面
26     @Override
27     public void onCreate(Bundle icicle) {
28         super.onCreate(icicle);
29         setContentView(R.layout.main);
30         mContext=this;
31         //验证当前存放股票代码的文件是否存在
32         File mFile =new File("/data/data/com.supermario.stocker/
           files/symbols.txt");
33         if(mFile.exists())
34         {
35             Log.e("guojs","file exist");
36         }else{
37             try {
38                 //新建存放股票代码的文件
39                 FileOutputStream outputStream=openFileOutput("symbols.
                   txt",MODE_PRIVATE);
40                 outputStream.close();
41             } catch (IOException e) {
42                 // TODO Auto-generated catch block
43                 e.printStackTrace();
44             }
45             Log.e("guojs","file no exist");
46         }
47         //初始化股票代码处理类
48         mDataHandler = new DataHandler(mContext);
49         //如果 adapter 数据为空显示的内容
50         getListView().setEmptyView(findViewById(R.id.empty));
51         quoteAdaptor = new QuoteAdaptor(this, this,mDataHandler);
52         //为列表设置适配器
53         this.setListAdapter(quoteAdaptor);
54         //添加股票按钮
```

```
55          addButton = (Button) findViewById(R.id.add symbols button);
56          //设置添加按钮监听器
57          addButton.setOnClickListener(this);
58          //股票输入文本框
59          symbolText= (EditText) findViewById(R.id.stock_symbols);
60      }
61      //生命周期：onCreate→onStart→onResume
62      protected void onResume(){
63          super.onResume();
64          if(quoteAdaptor != null)
65          {
66              //开始更新界面
67              quoteAdaptor.startRefresh();
68          }
69      }
70      //界面不可见时，停止更新
71      protected void onStop(){
72          super.onStop();
73          //停止更新界面
74          quoteAdaptor.stopRefresh();
75      }
```

12.3.3　适配器类 QuoteAdapter

股票数据列表相应的适配器类 QuoteAdapter 代码如下所示，该类中用于显示界面的核心函数为 getView，在该函数中使用 inflate 膨胀出界面元素的布局 R.layout.quote_cell，通过判断当前元素的奇偶性来决定当前元素的背景颜色，以示区分，如代码 046~050 行所示。

依次取得界面 quote_cell 中的各个 TextView 元素并分别赋值，其中在处理股票价格的时候需要注意数据的格式。如代码 066~087 行所示，需要将股票的价格通过 format 函数格式化成“#0.00”这种形式，并且为了让用户更好地区分股票当前是涨还是跌，在当前股票价格超过昨日收盘价时，将涨跌幅的字体设置成红色表示涨，反之设置字体颜色为绿色表示跌。

该适配器中还实现了股票的更新类 QuoteRefreshTask，如代码 144~170 行所示，设置每隔 10 秒更新一次数据。

```
001     //股票数据适配器
002     public class QuoteAdaptor extends BaseAdapter implements ListAdapter,
        Runnable {
003         //当前显示的最大数量为 10
004         private static final int DISPLAY_COUNT = 10;
005         public DataHandler dataHandler;
006         //强制更新标志
007         private boolean forceUpdate = false;
008         //保存上下文
009         Context context;
010         //保存 Activity 实例
011         Stocker stocker;
012         LayoutInflater inflater;
013
014         QuoteRefreshTask quoteRefreshTask = null;
015         int progressInterval;
016         //消息处理器
017         Handler messageHandler = new Handler();
```

```
018
019
020         public QuoteAdaptor(Stocker aController, Context mContext,
            DataHandler mdataHandler) {
021             //保存当前的上下文和 Activity 实例
022             context = mContext;
023             stocker = aController;
024             dataHandler = mdataHandler;
025         }
026         //取得股票数组的大小
027         public int getCount() {
028             return dataHandler.stocksSize();
029         }
030         //取得当前位置股票的对象
031         public StockInfo getItem(int position) {
032             return dataHandler.getQuoteForIndex(position);
033         }
034         //取得当前的位置
035         public long getItemId(int position) {
036             return position;
037         }
038         //生成视图
039         public View getView(int position, View convertView, ViewGroup
            parent) {
040             StockInfo quote;
041             inflater = LayoutInflater.from(context);
042             RelativeLayout cellLayout = (RelativeLayout)inflater.
                inflate(R.layout.quote cell, null);
043             cellLayout.setMinimumWidth(parent.getWidth());
044             int color;
045             stocker.setProgress(progressInterval*(position + 1));
046             if(position % 2 > 0)
047                 color = Color.rgb(48,92,131);
048             else
049                 color = Color.rgb(119,138,170);
050             cellLayout.setBackgroundColor(color);
051             quote = dataHandler.getQuoteForIndex(position);
052             TextView field = (TextView)cellLayout.findViewById
               (R.id.symbol);
053             //设置股票的代码
054             field.setText(quote.getNo());
055             field.setClickable(true);
056             field.setOnClickListener(stocker);
057
058             //股票名字
059             field = (TextView)cellLayout.findViewById(R.id.name);
060             field.setClickable(true);
061             field.setOnClickListener(stocker);
062             field.setText(quote.getName());
063
064             field = (TextView)cellLayout.findViewById(R.id.current);
065             //设置股票当前价格
066             double current=Double.parseDouble
                (quote.getCurrent price());
067             double closing price=Double.parseDouble
                (quote.getClosing price());
068             //保留两位小数
069             DecimalFormat df=new DecimalFormat("#0.00");
070             String percent=df.format(((current-closing_price)*100/
                closing_price))+"%";
```

```
071             field.setText(df.format(current));
072             field.setClickable(true);
073             field.setOnClickListener(stocker);
074
075             field = (TextView)cellLayout.findViewById(R.id.percent);
076             //若股票价格超过昨日收盘价
077             if(current > closing_price)
078             {
079                 //设置字体颜色为红色
080                 field.setTextColor(0xffEE3B3B);
081             }
082             else
083             {
084                 //设置字体颜色为绿色
085                 field.setTextColor(0xff2e8b57);
086             }
087             field.setText(percent);
088             cellLayout.setId(position + 33);
089             cellLayout.setClickable(true);
090             cellLayout.setOnClickListener(stocker);
091             return cellLayout;
092         }
093         //所有元素均可选中
094         public boolean areAllItemsSelectable() {
095             return true;
096         }
097         public boolean isSelectable(int arg0) {
098             return true;
099         }
100         //停止更新股票
101         public void stopRefresh(){
102             quoteRefreshTask.cancelTimer();
103             quoteRefreshTask = null;
104         }
105         //开始更新股票
106         public void startRefresh(){
107             if(quoteRefreshTask == null)
108                 quoteRefreshTask = new QuoteRefreshTask(this);
109         }
110         //更新适配器
111         public void refreshQuotes(){
112             messageHandler.post(this);
113         }
114         //更新适配器内容
115         public void run(){
116             if(mDataHandler.stocksSize() > 0){
117                 if(forceUpdate ){
118                     forceUpdate = false;
119                     progressInterval = 10000/DISPLAY_COUNT;
120                     stocker.setProgressBarVisibility(true);
121                     stocker.setProgress(progressInterval);
122                     mDataHandler.refreshStocks();
123                 }
124                 //通知数据更改
125                 this.notifyDataSetChanged();
126             }
127         }
128         //添加股票代码到文件中
129         public void addSymbolsToFile(ArrayList<String> symbols){
```

```
130             //强行更新页面数据
131             forceUpdate = true;
132             //添加股票到文件中
133             mDataHandler.addSymbolsToFile(symbols);
134             //添加消息到消息队列
135             messageHandler.post(this);
136         }
137         //移除列表中的数据
138         public void removeQuoteAtIndex(int index){
139             forceUpdate = true;
140             mDataHandler.removeQuoteByIndex(index);
141             messageHandler.post(this);
142         }
143         //股票更新定时器
144         public class QuoteRefreshTask extends TimerTask {
145             QuoteAdaptor quoteAdaptor;
146             Timer refreshTimer;
147             final static int  TENSECONDS = 10000;
148             public QuoteRefreshTask(QuoteAdaptor anAdaptor){
149                 refreshTimer = new Timer("Quote Refresh Timer");
150                 refreshTimer.schedule(this, TENSECONDS, TENSECONDS);
151                 quoteAdaptor = anAdaptor;
152             }
153 
154             public void run(){
155                 messageHandler.post(quoteAdaptor);
156             }
157 
158             public void startTimer(){
159                 if(refreshTimer == null){
160                     refreshTimer = new Timer("Quote Refresh Timer");
161                     refreshTimer.schedule(this, TENSECONDS, TENSECONDS);
162                 }
163             }
164             //取消定时器
165             public void cancelTimer(){
166                 this.cancel();
167                 refreshTimer.cancel();
168                 refreshTimer = null;
169             }
170         }
171     }
```

12.3.4　设置按键监听器

接着要为界面设置一些按键监听器，代码如下所示。当单击界面元素时调用 onListItemClick()函数，在该函数中将生成一个对话框，并将该对话框的布局设置为 quote_detail，接下来将获得的 StockInfo 对象的内容依次填充到界面中，如图 12.2 所示。

若用户需要添加股票，需要先在股票代码输入框中输入股票代码如“sh601007”，接着单击“添加”按钮执行 addSymbol()函数。在该函数中将输入的字符串进行处理，以空格将字符串分割成字符串数组，最后将数组中的股票代码分别添加到股票代码文件 symnols.txt 中。



图 12.2　查看股票详细信息

```
001 //列表元素被单击之后触发
002 protected void onListItemClick(ListView l, View v, int position, longid){
003     super.onListItemClick(l,v, position, id);
004     //取得单击位置的股票
005     StockInfo quote = quoteAdaptor.getItem(position);
006     //取得当前位置的序号
007     currentSelectedIndex=position;
008     if(dialog == null){
009         dialog = new Dialog(mContext);
010         dialog.setContentView(R.layout.quote_detail);
011         //"删除"按钮
012         deleteButton = (Button) dialog.findViewById(R.id.delete);
013         //设置“删除”按钮监听器
014         deleteButton.setOnClickListener(this);
015         //"返回"主界面按钮
016         cancelButton = (Button) dialog.findViewById(R.id.close);
017         //设置"返回"按钮监听器
018         cancelButton.setOnClickListener(this);
019         //当前股票价格
020         currentTextView = (TextView) dialog.findViewById(R.id.current);
021         //当前股票编码
022         noTextView = (TextView) dialog.findViewById(R.id.no);
023         //昨日收盘价
024         openTextView = (TextView) dialog.findViewById
            (R.id.opening_price);
025         //今日收盘价
026         closeTextView = (TextView) dialog.findViewById
            (R.id.closing price);
027         //今日最低价
028         dayLowTextView = (TextView) dialog.findViewById(R.id.day_low);
029         //今日最高价
030         dayHighTextView = (TextView) dialog. findViewById
            (R.id.day high);
031         //股票 K 线图
032         chartView = (ImageView)dialog.findViewById(R.id.chart_view);
033     }
034     //设置对话框标题
```

```
035     dialog.setTitle(quote.getName());
036     //设置股票当前价格
037     double current=Double.parseDouble(quote.getCurrent price());
038     double closing_price=Double. parseDouble(quote.
        getClosing_price());
039     //保留两位小数
040     DecimalFormat df=new DecimalFormat("#0.00");
041     String percent=df.format
        (((current-closing_price)*100/closing_price))+"%";
042     //若股票价格超过昨日收盘价
043     if(current > closing_price)
044     {
045         //设置字体颜色为红色
046         currentTextView.setTextColor(0xffEE3B3B);
047     }
048     else
049     {
050         //设置字体颜色为绿色
051         currentTextView.setTextColor(0xff2e8b57);
052     }
053     //设置 TextView 内容
054     currentTextView.setText(df.format(current)+"  ("+percent+")");
055     openTextView.setText(quote.opening price);
056     closeTextView.setText(quote.closing price);
057     dayLowTextView.setText(quote.min price);
058     dayHighTextView.setText(quote.max price);
059     noTextView.setText(quote.no);
060     //设置 K 线图
061  chartView.setImageBitmap(mDataHandler.getChartForSymbol(quote.no));
062     dialog.show();
063 }
064 //判断回车键按下时添加股票
065 public boolean onKeyUp(int keyCode, KeyEvent event){
066     if(keyCode == KeyEvent.KEYCODE ENTER){
067         //添加股票
068         addSymbol();
069         return true;
070     }
071     return false;
072 }
073 //添加股票代码，以空格或者回车分隔多个股票
074 private void addSymbol(){
075     //获得文本框的输入内容
076     String symbolsStr = symbolText.getText().toString();
077     //将回车符替换成空格
078     symbolsStr = symbolsStr.replace("\n", " ");
079     //以空格分割字符串
080     String symbolArray[] = symbolsStr.split(" ");
081     int index, count = symbolArray.length;
082     ArrayList<String> symbolList = new ArrayList<String>();
083     for(index = 0; index < count; index++){
084         symbolList.add(symbolArray[index]);
085     }
086     //将股票代码添加进文件中
087     quoteAdaptor.addSymbolsToFile(symbolList);
088     //设置文本框为空
089     symbolText.setText(null);
090 }
091 //设置按键回调函数
```

```
092 public void onClick(View view) {
093     if(view == addButton){
094         //添加股票到文件中
095         addSymbol();
096     } else if(view == cancelButton){
097         //关闭对话框
098         dialog.dismiss();
099     } else if(view == deleteButton){
100         //删除当前股票
101         quoteAdaptor.removeQuoteAtIndex(currentSelectedIndex);
102         dialog.dismiss();
103     } else if(view.getParent() instanceof RelativeLayout){
104         RelativeLayout rl = (RelativeLayout)view.getParent();
105         this.onListItemClick(getListView(), rl, rl.getId()-33,rl.
            getId());
106     } else if(view instanceof RelativeLayout){
107         this.onListItemClick(getListView(), view, view.getId()-33,
            view.getId());
108     }
109 }
```

12.3.5　数据查询

看完了整个主界面执行的流程，我们来分析一下主界面用到的数据处理类 DataHandler。在这个类中需要定义几个比较重要的变量，如股票信息查询网址 QUERY_URL、K 线图网址、股票代码存放文件 SYMBOL_FILE_NAME 等，这些都是数据查询的基础。接着在构造函数中执行 readStockFromFile，读取文件中的股票代码信息存放到数组 stocks 中，并调用函数 refreshStocks 更新股票数据。

```
01 //股票数据处理类
02 public class DataHandler {
03     private static final String TAG="DataHandler";
04     //股票查询网址
05     private static final String QUERY URL = "http://hq.sinajs.cn/list=";
06     //股票 K 线图
07     private static final String QUERY_IMG ="http:
       //image.sinajs.cn/newchart/daily/n/";
08     //存储股票代码的文件
09     private static final String SYMBOL FILE NAME = "symbols.txt";
10     private int BUF SIZE = 16384;
11     //存储股票代码的数组
12     private ArrayList<String> stocks;
13     //存储股票数据的数组
14     private ArrayList<StockInfo> stockInfo=new ArrayList<StockInfo>();
15     //各个股票信息在数组中的索引
16     private final int NAME=0;
17     private final int OPENING PRICE=1;
18     private final int CLOSING PRICE=2;
19     private final int CURRENT_PRICE=3;
20     private final int MAX_PRICE=4;
21     private final int MIN_PRICE=5;
22     Context context;
23     public DataHandler(Context mContext){
24         super();
25         //读取存储在文件中的股票代码信息
```

```
26          readStockFromFile();
27          context=mContext;
28          if(stocks != null)
29          {
30              //更新股票数据
31              refreshStocks();
32          }
33      }
```

12.3.6　读取股票代码信息

读取股票代码信息的函数实现代码如下所示，取出文件 symbols.txt 中的信息存放到字符串中，接着以“,”分割字符串成数组，数组中的数据即为股票的数据，最后将该数组的数据存放到 stocks 中。

该类还实现了股票的添加操作，如下函数 addSymbolsToFile()所示，添加之前首先判断要添加的股票代码是否已存在数组 stocks 中，若不存在则添加到数组 newStocks 中。接着执行 _addQuotes(newStocks) 尝试去获取指定股票的数据，并调用函数 parseQuotesFromStream()对数据进行解析。如果能解析到想要的数据，说明该股票的代码是有效的，否则说明是无效的。若股票的代码有效，则将股票代码添加到 stocks 数组中。最后执行 savePortfolio 将 stocks 的信息重新写入到文件中，实现股票代码的保存。

```
001 //读取存储在文件中的股票代码信息
002 private void readStockFromFile(){
003     File fullPath;
004     if(stocks == null)
005     {
006         //初始化股票代码数组
007         stocks = new ArrayList<String>();
008     }
009     FileInputStream inStream;
010     BufferedReader bReader;
011     String quoteStr="";
012     //获取存储股票的文件
013     fullPath = new File("/data/data/com. supermario.stocker/
        files/symbols.txt");
014         //读取文件
015     try {
016         inStream = new FileInputStream(fullPath);
017         bReader = new BufferedReader(new InputStreamReader(inStream));
018         quoteStr = bReader.readLine();
019         bReader.close();
020         inStream.close();
021     } catch (IOException e) {
022         // TODO Auto-generated catch block
023         e.printStackTrace();
024     }
025     //如果文件中不包含股票代码数据
026     if(quoteStr == "" || quoteStr == null)
027     {
028         //数组清空
029         stocks.clear();
030         return;
031     }
032     //将字符串切割成数组
```

```
033     String strArray[] = quoteStr.split(",");
034     int index, count = strArray.length;
035     //重置数组
036     stocks.clear();
037     //将所有股票添加到数组中
038     for(index = 0; index < count; index++)
039         stocks.add(strArray[index]);
040 }
041 //添加股票代码到文件中
042 public synchronized void addSymbolsToFile(ArrayList<String>stockList){
043     if(stockList != null){
044         //如果股票数组中没有数据
045         if(stocks == null || stocks.size() == 0){
046             //直接将参数添加到股票数组中
047             stocks = new ArrayList<String>();
048             stocks.addAll(stockList);
049         } else {
050             int i1, i2;
051             //文件中股票数组大小
052             int c1 = stocks.size();
053             //欲添加的股票数组大小
054             int c2 = stockList.size();
055             ArrayList<String> newStocks = new ArrayList<String>();
056             //判断欲添加的股票代码是否在原来的数组中
057             boolean foundSymbol = false;
058             //循环遍历，查找出新股票
059             for(i2 = 0; i2 < c2; i2++){
060                 String newSymbol = stockList.get(i2);
061                 for(i1 = 0; i1 < c1; i1++){
062                     if(newSymbol.equals(stocks.get(i1))){
063                         foundSymbol = true;
064                         break;
065                     }
066                 }
067                 //若为新股票，则添加进新股票数组中
068                 if(!foundSymbol){
069                     newStocks.add(newSymbol);
070                 }
071             }
072             if(newStocks.size() > 0){
073                  addQuotes(newStocks);
074             }
075         }
076         //保存股票
077         savePortfolio();
078     }
079 }
080 //根据股票代码数组添加股票
081 protected void  addQuotes(ArrayList<String> stockSymbols){
082     if(stockSymbols != null && stockSymbols.size() > 0){
083         int index, count = stockSymbols.size();
084         //取得 http 客户端实例
085         HttpClient req = new DefaultHttpClient();
086         //用于存放网址
087         StringBuffer buf = new StringBuffer();
088         //构建网址
089         buf.append(QUERY_URL);
090         buf.append(stockSymbols.get(0));
091         //如果股票数超过 1
```

```
092         for(index = 1; index < count; index++){
093             buf.append(",");
094             buf.append(stockSymbols.get(index));
095         }
096         try {
097             //采用 get 方式获取网页数据
098             HttpGet httpGet = new HttpGet(buf.toString());
099             HttpResponse response = req.execute(httpGet);
100             InputStream iStream = response.getEntity().getContent();
101             //解析网页数据，若解析成功则添加股票
102             if(parseQuotesFromStream(iStream))
103             {
104                 for(index = 0; index < count; index++){
105                     stocks.add(stockSymbols.get(index));
106                 }
107                 Toast.makeText(context, "股票添加成功！", Toast.
                    LENGTH_SHORT).show();
108             }else{
109                 Toast.makeText(context, "股票添加失败！", Toast.
                    LENGTH_SHORT).show();
110             }
111         } catch (IOException e){
112             Log.e(TAG, e.getMessage());
113         }
114     }
115 }
116 //将股票添加进文件中
117 private void savePortfolio(){
118     //将当前 stocks 中的值重新写入文件中
119     if(stocks.size() > 0){
120         FileOutputStream outStream = null;
121         OutputStreamWriter oWriter;
122         try {
123             //打开文件
124             outStream = context.openFileOutput(SYMBOL_FILE_NAME,
                Context.MODE_PRIVATE);
125             oWriter = new OutputStreamWriter(outStream);
126             StringBuffer buf = new StringBuffer();
127             int index, count = stocks.size();
128             //构建字符串写入文件中
129             buf.append(stocks.get(0));
130             for(index = 1; index < count; index++){
131                 buf.append(",");
132                 buf.append(stocks.get(index));
133             }
134             String outStr = buf.toString();
135             oWriter.write(outStr, 0, outStr.length());
136             oWriter.close();
137             outStream.close();
138         } catch(IOException e){
139             Log.e(TAG, e.getMessage());
140         }
141     }
142 }
143 //解析股票数据
144 private boolean parseQuotesFromStream(InputStream aStream){
145     boolean flag=false;
146     if(aStream != null){
```

```
147         stockInfo.clear();
148         //读取数据
149         BufferedInputStream iStream = new BufferedInputStream(aStream);
150         InputStreamReader iReader=null;
151         try {
152             //使用 GBK 方式解析数据
153             iReader = new InputStreamReader(iStream,"GBK");
154         } catch (UnsupportedEncodingException e) {
155             // TODO Auto-generated catch block
156             e.printStackTrace();
157         }
158         StringBuffer strBuf = new StringBuffer();
159         char buf[] = new char[BUF SIZE];
160         try {
161             int charsRead;
162             //将数据读取到 StringBuffer 中
163             while((charsRead = iReader.read(buf, 0, buf.length)) != -1){
164                 strBuf.append(buf, 0, charsRead);
165             }
166             //匹配股票数据
167             Pattern pattern=Pattern.compile("str_(.+)=\"(.+)\"");
168             Matcher matcher=pattern.matcher(strBuf);
169             while(matcher.find()){
170                 //股票信息为第二个括号中对应的内容
171                 String result=matcher.group(2);
172                 String[] data=result.split(",");
173                 StockInfo mStockInfo=new StockInfo();
174                 //存储股票的代码
175                 mStockInfo.setNo(matcher.group(1));
176                 //存储股票名字
177                 mStockInfo.setName(data[NAME]);
178                 //存储股票今日开盘价
179                 mStockInfo.setOpening price(data[OPENING PRICE]);
180                 //存储股票昨日收盘价
181                 mStockInfo.setClosing price(data[CLOSING PRICE]);
182                 //存储股票当前价格
183                 mStockInfo.setCurrent_price(Double.parseDouble
                    (data[CURRENT_PRICE]) +0.01*(int)(10*Math.random())+"");
184                 //存储股票今日最高价格
185                 mStockInfo.setMax price(data[MAX PRICE]);
186                 //存储股票今日最低价格
187                 mStockInfo.setMin_price(data[MIN_PRICE]);
188                 //将数据存储到数组中
189                 stockInfo.add(mStockInfo);
190                 flag=true;
191             }
192         }catch(IOException iox){
193             Log.e(TAG, iox.getMessage());
194         }
195     }
196     return flag;
197 }
```

12.3.7　股票的更新

股票信息的更新函数 refreshStocks()如以下代码 63～71 行所示，在该函数中主要调用

getQuotesForArray 进行股票信息的更新，函数 getQuotesForArray()通过解析数组 stocks 中对应的股票信息，并将信息存储到 stockInfo 中返回来达到更新的目的。

除了股票更新函数之外，本类还实现了获得指定股票代码的日 K 线图，如函数 getChartForSymbol()所示，以及取得执行索引的股票信息 getQuoteForIndex 和删除执行索引的股票信息 removeQuoteByIndex。

```
01  //取得存储股票数据的数组
02   protected  ArrayList<StockInfo>  getQuotesForArray(ArrayList<String>
     stockSymbols){
03     if(stockSymbols != null && stockSymbols.size() > 0){
04         //取得 http 客户端实例
05         HttpClient req = new DefaultHttpClient();
06         //用于存放网址
07         StringBuffer buf = new StringBuffer();
08         int index;
09         //取得股票的总数
10         int count = stockSymbols.size();
11         //构建网址
12         buf.append(QUERY URL);
13         buf.append(stockSymbols.get(0));
14         //如果股票数超过 1
15         for(index = 1; index < count; index++){
16             buf.append(",");
17             buf.append(stockSymbols.get(index));
18         }
19         try {
20             //采用 get 方式获取网页数据
21             HttpGet httpGet = new HttpGet(buf.toString());
22             HttpResponse response = req.execute(httpGet);
23             InputStream iStream = response.getEntity().getContent();
24             //解析网页数据
25             parseQuotesFromStream(iStream);
26             //返回股票数据
27             return stockInfo;
28         } catch (IOException e){
29             Log.e(TAG, e.getMessage());
30         }
31         return null;
32     }
33     return null;
34  }
35  //取得 K 线图
36  public Bitmap getChartForSymbol(String symbol){
37     try {
38         try {
39             //构建股票 K 线图网址
40             StringBuilder sb = new StringBuilder(QUERY IMG);
41             sb = sb.append(symbol+".gif");
42             HttpClient req = new DefaultHttpClient();
43             HttpGet httpGet = new HttpGet(sb.toString());
44             //访问执行网址
45             HttpResponse response = req.execute(httpGet);
46             InputStream iStream;
47             BitmapDrawable bitMap;
48             //取得网址返回的内容
49             iStream = response.getEntity().getContent();
50             //将返回的内容解析成图片
```

```
51              bitMap = new BitmapDrawable(iStream);
52              iStream.close();
53              iStream = null;
54              return bitMap.getBitmap();
55          } catch ( IOException iox ){
56              Log.d(TAG, iox.getMessage());
57          }
58      } catch (Exception e) {
59          Log.d(TAG, e.getMessage());
60      }
61      return null;
62  }
63  //更新股票数据
64  public  void refreshStocks(){
65      long startTime = System.currentTimeMillis();
66      long endTime;
67      //取得股票的最新数据
68      getQuotesForArray(stocks);
69      endTime = System.currentTimeMillis();
70      Log.d(TAG, "Refresh ran for " + (endTime - startTime) + "milli
        senconds");
71  }
72  //返回股票数组大小
73  public  int stocksSize(){
74      if(stocks != null)
75            return stocks.size();
76      return 0;
77  }
78  //通过股票的索引删除股票
79  public  void removeQuoteByIndex(int index){
80      stocks.remove(index);
81      //保存当前股票
82      savePortfolio();
83  }
84  //返回指定位置的数组元素
85  public  StockInfo getQuoteForIndex(int index){
86      return stockInfo.get(index);
87  }
```

12.4　知 识 拓 展

这次要给大家介绍的是 openFileOutput()函数的使用，Activity 提供了 openFileOutput()方法可以把数据输出到文件中，如下代码所示：

```
01  public void save()
02      {
03          try {
04              FileOutputStream outStream=this.openFileOutput
                ("a.txt",Context.MODE_WORLD_READABLE);
05              outStream.write(text.getText().toString().getBytes());
06              outStream.close();
07              Toast.makeText(MyActivity.this,"Saved",Toast.LENGTH_ LONG)
                .show();
08          } catch (FileNotFoundException e) {
09              return;
10          }
```

```
11          catch (IOException e){
12              return ;
13          }
14      }
```

openFileOutput()方法的第一参数用于指定文件名称，不能包含路径分隔符“/”，如果文件不存在，Android 会自动创建它。创建的文件保存在/data/data/<package name>/files 目录，如/data/data/cn.itcast.action/files/itcast.txt。通过单击 Eclipse 菜单 Window|Show View|Other 命令，在对话窗口中展开 android 文件夹，选择下面的 File Explorer 视图，然后在 File Explorer 视图中展开/data/data/<package name>/files 目录就可以看到该文件。

openFileOutput()方法的第二参数用于指定操作模式，有 4 种模式，分别为：

- Context.MODE_PRIVATE=0
- Context.MODE_APPEND=32768
- Context.MODE_WORLD_READABLE=1
- Context.MODE_WORLD_WRITEABLE =2

这 4 中模式分别说明如下：

- Context.MODE_PRIVATE：为默认操作模式，代表该文件是私有数据，只能被应用本身访问，在该模式下，写入的内容会覆盖原文件的内容，如果想把新写入的内容追加到原文件中，可以使用 Context.MODE_APPEND。
- Context.MODE_APPEND：该模式会检查文件是否存在，存在就往文件追加内容，否则就创建新文件。
- Context.MODE_WORLD_READABLE 和 Context.MODE_WORLD_WRITEABLE：用来控制其他应用是否有权限读写该文件。
- MODE_WORLD_READABLE：表示当前文件可以被其他应用读取。
- MODE_WORLD_WRITEABLE：表示当前文件可以被其他应用写入。

如果希望文件被其他应用读和写，可以传入 openFileOutput(“itcast.txt”, Context.MODE_WORLD_READABLE + Context.MODE_WORLD_WRITEABLE)。

Android 有一套自己的安全模型，应用程序（.apk）在安装时系统就会分配给它一个 userid，当该应用要去访问其他资源比如文件的时候，就需要 userid 匹配。默认情况下，任何应用创建的文件，如 sharedpreferences，数据库都应该是私有的（位于/data/data/<package name>/files），其他程序无法访问。除非在创建时指定了 Context.MODE_WORLD_READABLE 或者 Context.MODE_WORLD_WRITEABLE，只有这样其他程序才能正确访问。

12.5　本 章 小 结

本章介绍了如何开发一个实时查看股票信息的软件，主要实现了股票的添加、查看、删除等功能。本章需要重点掌握的知识是如何实时更新从网络中获得的数据，如何对这些数据进行筛选和存储，如何组织布局将数据呈现给用户等。

第 13 章　Google 天气客户端

大家在使用 Android 手机的时候，肯定都用过天气预报的应用，现在网上已经有不少成熟的产品。当然，作为开发者而言，一定会对这种应用的开发很感兴趣，我们能不能自己来写一款类似的应用？答案是肯定的，而且非常简单。

13.1　功 能 分 析

首先，要开发一款天气预报应用，一定要有一个 Web 服务端来提供数据，这个数据源我们自己肯定是没办法弄的，所以就需要一个第三方机构为我们提供天气数据。这种机构其实有很多，不过大多数都是收费的，当然这些收费的数据源提供的数据会更加丰富详细。如果不想花钱去购买这些收费的数据服务，我们还有另一种替代方案——就是使用免费的天气数据。

Google 天气预报就是其中一种免费的天气预报，也是我们这个程序开发的基础。Google 开发了一套天气预报的 API，可以使用多种方式进行查询：

（1）使用邮政编码（美国）：http://www.google.com/ig/api?hl=zh-cn&weather=94043（加州山景城）。

（2）使用经度纬度坐标：http://www.google.com/ig/api?hl=zh-cn&weather=,,,30670000,104019996（成都）。

（3）使用通行城市名称：

http://www.google.com/ig/api?weather=Beijing&hl=zh-cn（北京）

http://www.google.com/ig/api?weather=Osaka&hl=zh-cn（大阪）

或

http://www.google.com/ig/api?weather=Beijing&hl=zh（北京）

http://www.google.com/ig/api?weather=Osaka&hl=ja（大阪）

可以查找到哪些国家和城市呢？Google 也提供了接口，返回的类型也可以根据 output 参数来指定：

（1）查找国家 http://www.google.com/ig/countries?output=xml&hl=zh-cn（返回 xml）。

（2）查找城市 http://www.google.com/ig/cities?hl=zh-cn&country=cn（返回 json）。

有了这些数据，在自己的应用里加入天气预报就不难了。下面先看下效果图，如图 13.1 所示。

通过效果图可以看出，整个界面布局比较简单。提供两种方式进行查询：一种是从预

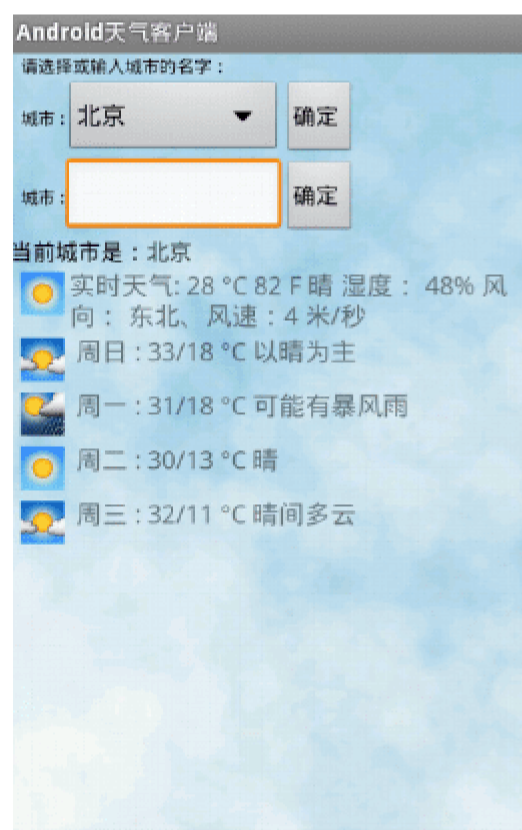


图 13.1　Android 天气预报客户端

定义的城市列表中选择查看相应城市的天气预报信息，一种是用户自己输入城市名称，进行查询。查询结果分为两种，一种是实时天气信息，一种是天气预报信息。

通过以上分析我们可以看出，整个程序的核心就是解析 XML 文件，Android 提供了 3 种方式用于解析 XML 文件：DOM 解析、SAX 解析、PULL 解析。

13.2　XML 解析

在开发这个程序之前我们有必要先熟悉一下 android XML 的解析方式，下面我们分别介绍一下这 3 种解析方式。

13.2.1　DOM 解析

DOM 解析 XML 文件时，会将 XML 文件的所有内容读取到内存中，然后允许使用 DOM API 遍历 XML 树、检索所需的数据。使用 DOM 操作 XML 的代码看起来比较直观，并且，在某些方面比基于 SAX 的实现更加简单。但是，因为 DOM 需要将 XML 文件的所有内容读取到内存中，所以内存的消耗比较大，特别对于运行 Android 的移动设备来说，因为设备的资源比较宝贵，所以建议还是采用 SAX 来解析 XML 文件。当然，如果 XML 文件的内容比较小，采用 DOM 是可行的。

（1）我们用一个简单的例子来学习一下 DOM 解析的原理，XML 文件如下：

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <resources>
3       <fruit id="1" name="apple" ><fruit id="01" name="grap">
         10yuan</fruit></fruit>
4       <fruit id="2" name="banana" >
5        3yuan
6       </fruit>
7       <fruit id="3" name="pear" />
8   </resources>
```

（2）在主界面中简单地添加一个 Button 和 TextView，Button 用来响应单击事件，

TextView 用来显示结果。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.
    android.com/apk/res/android"
03      android:layout_width="fill_parent"
04      android:layout_height="fill_parent"
05      android:orientation="vertical" >
06      <!--添加一个按钮  -->
07      <Button
08          android:id="@+id/start"
09          android:layout width="wrap content"
10          android:layout height="wrap content"
11          android:text="GO!" />
12      <!-- 添加一个文本框，用于显示结果 -->
13      <TextView
14          android:id="@+id/show"
15          android:layout width="fill parent"
16          android:layout height="wrap content"
17          android:text="" />
18  </LinearLayout>
```

（3）接下去在 DomXMLActivity 中，先初始化一些界面元素，并为按钮绑定监听事件。当按下按键的时候，开始解析 Assets 中的 fruit.xml 文件，并将结果显示到 TextView 中。解析过程中，我们需要先实例化一个 DocumentBuilderFactory 类，使得程序可以从文件流中初始化 DOM 对象树，如以下代码第 60 行所示。然后通过这个 DocumentBuilderFactory 去实例化一个 DocumentBuilder，使用这个 DocumentBuilder 去解析 XML 文件流。

这里可以把 DocumentBuilderFactory 比作一个工厂，而 DocumentBuilder 是工厂的机器。整个流程可以理解为，在一个专门使用 DOM 解析 XML 的工厂里面，用一个机器采用 DOM 的方式去解析 XML。其实 Java 的这种架构还是挺有意思的，很贴合实际。

```
01  package com.supermario.domxml;          //声明包语句
02~17 行为引入相关类，这里不再列举，请阅读光盘内容
//……
18  public class DomXMLActivity extends Activity {
19      //新建一个按钮用于响应用户按键
20      private Button start;
21      //新建一个 TextView 用于存放结果
22      private TextView show;
23      //Assets 中的 xml 文件名称
24      private String fileName="fruit.xml";
25      InputStream inStream=null;
26      /** 首次创建界面时运行*/
27      @Override
28      public void onCreate(Bundle savedInstanceState)  {
29          super.onCreate(savedInstanceState);
30          setContentView(R.layout.main);
31          show=(TextView)findViewById(R.id.show);
32          start=(Button)findViewById(R.id.start);
33
34          try {
35              //从 Assets 中获取文件
36              inStream = this.getAssets().open(fileName);
37          } catch (IOException e)  {
38              // TODO Auto-generated catch block
39              e.printStackTrace();
```

```
40          }
41          //为按键绑定事件
42          start.setOnClickListener(new OnClickListener(){
43              @Override
44              public void onClick(View v) {
45                  // TODO Auto-generated method stub
46                  //用于存放结果字符串
47                  String result="";
48                  //解析字符流
49                  result=parse(inStream);
50                  //将结果显示到界面上
51                  show.setText(result);
52              }
53          });
54      }
55      //解析字符流
56      public String parse(InputStream inStream)
57      {
58          String result="";
59          //实例化一个 DocumentBuilderFactory 类
60            DocumentBuilderFactory dbf=DocumentBuilder
        Factory.newInstance();
61          DocumentBuilder builder=null;
62          Document doc=null;
63          try {
64              //实例化一个 DocumentBuilder 用于解析字符流
65              builder=dbf.newDocumentBuilder();
66          } catch (ParserConfigurationException e) {
67              // TODO Auto-generated catch block
68              e.printStackTrace();
69          }
70          try {
71              //解析字符流
72              doc=builder.parse(inStream);
73          } catch (SAXException e) {
74              // TODO Auto-generated catch block
75              e.printStackTrace();
76          } catch (IOException e) {
77              // TODO Auto-generated catch block
78              e.printStackTrace();
79          }
80          Element ele=doc.getDocumentElement();
81          //获取所有的 fruit 节点
82          NodeList nl=ele.getElementsByTagName("fruit");
83          if(nl != null && nl.getLength() != 0)
84          {
85              for(int i=0;i<nl.getLength();i++)
86              {
87                  Element entry=(Element)nl.item(i);
88                  //用于获取属性
89                  result += "name:"+entry.getAttribute("name")+"-->";
90                  //用于获取文本内容
91                  result += entry.getTextContent()+"\n";
92              }
93          }
94          return result;
95      }
96  }
```

（4）实际运行效果如图 13.2 所示。可以看到解析的时候，节点内容里面的换行符也被解析出来了，这一点我们在作处理的时候不能忽略了。



图 13.2　DOM 解析 XML

13.2.2　SAX 解析

SAX 是一种占用内存少且解析速度快的解析器，它采用的是事件驱动，它不需要解析完整个文档，而是按照内容顺序，看文档某个部分是否符合 xml 语法，如果符合就触发相应的事件。所谓的事件就是些回调方法（callback），这些方法定义在 ContentHandler 中，下面是其主要方法：

- startDocument：当遇到文档的时候就触发这个事件调用这个方法 可以在其中做些预处理工作。
- startElement(String namespaceURI,String localName,String qName,Attributes atts)：当遇开始标签的时候就会触发这个方法。
- endElement（String uri,String localName,String name）：当遇到结束标签时触发这个事件，调用此方法可以做些善后工作。
- characters(char [] ch,int start,int length)：当遇到 xml 内容时触发这个方法。

下面我们同样以一个小例子来学习一下 SAX 解析的原理。

（1）首先新建一个工程 SaxXML，包名为 com.supermario.saxxml。然后在 assets 目录下新建一个 xml 文件 student.xml，用于存放一些待解析的信息。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <stundets>
03     <student id="20120812115">
04       <name>张三</name>
05       <speciality>通信工程</speciality>
06       <qq>843200157</qq>
07     </student>
08     <student id="20120812116">
09       <name>李四</name>
10       <speciality>网络工程</speciality>
11       <qq>812256156</qq>
12     </student>
```

```
13      <student id="20120812117">
14        <name>王五</name>
15        <speciality>软件工程</speciality>
16        <qq>812750158</qq>
17      </student>
18  </stundets>
```

（2）在包 com.supermario.saxxml 下新建一个类 Student.java，用于存放学生的信息。

```
01  public class Student {
02      long Id;                        //用于存放 id 信息
03      String Name;                    //用于存放 Name 信息
04      String Speciality;              //用于存放专业信息
05      long QQ;                        //用于存放 QQ 信息
06      //带参数构造函数，用于初始化类
07      public Student(long id, String name, String speciality, long qQ) {
08          super();
09          Id = id;
10          Name = name;
11          Speciality = speciality;
12          QQ = qQ;
13      }
14      //不带参数构造函数
15      public Student() {
16          super();
17      }
18      //取得 id
19      public long getId() {
20          return Id;
21      }
22      //取得 Name
23      public String getName() {
24          return Name;
25      }
26      //取得 QQ
27      public long getQQ() {
28          return QQ;
29      }
30      //取得专业信息
31      public String getSpeciality() {
32          return Speciality;
33      }
34      //设置 id
35      public void setId(long id) {
36          Id = id;
37      }
38      //设置姓名
39      public void setName(String name) {
40          Name = name;
41      }
42      //设置 QQ
43      public void setQQ(long qQ) {
44          QQ = qQ;
45      }
46      //设置专业
47      public void setSpeciality(String speciality) {
48          Speciality = speciality;
49      }
50  }
```

（3）在主界面布局文件 main.xml 中，创建一个 Button 和 ListView。在 SaxXMLActivity.java 中先初始化界面元素，为按钮绑定监听器。当单击 Button 按钮时，开始解析 xml 文件，如以下代码 39 行所示。经过解析之后得到一个由多个 student 元素组成的 list。为了方便显示，我们将这个 list 中的 student 元素转换成字符串，存储到另外一个 list 中。

```
01  package com.supermario.saxxml;            //声明包语句
02~16 行为引入相关类，这里不再列举，请阅读光盘内容
//……
17  public class SaxXMLActivity extends Activity {
18
19      //新建一个按键
20      private Button button;
21      //新建一个列表
22      private ListView listView;
23      //新建一个数组列表用于存放字符串数组
24      private ArrayList<String> list=new ArrayList<String>();
25      public void onCreate(Bundle savedInstanceState) {
26      super.onCreate(savedInstanceState);
27      setContentView(R.layout.main);
28      button=(Button)findViewById(R.id.btn1);
29      listView=(ListView) findViewById(R.id.listView1);
30      //为按键绑定监听器
31      button.setOnClickListener(new ButtonListener());
32      }
33
34      class ButtonListener implements OnClickListener{
35
36      @Override
37      public void onClick(View v) {
38         //将解析后的结果存储到 students 中
39         List<Student> students=parserXMl();
40         //枚举数组中的元素
41         for (Iterator iterator = students.iterator(); iterator.hasNext();) {
42             Student student = (Student) iterator.next();
43             //将类的内容转换成字符串，依次存储到 list 中
44             list.add(String.valueOf(student.getId())+" "+student.
              getName()+" "+student.getSpeciality()+" "+String.valueOf
              ((student.getQQ())));
45         }
46         //新建一个适配器 adaapter 用于给 listview 提供数据
47         ArrayAdapter<String> adapter=new ArrayAdapter<String> (SaxXML
          Activity.this, android.R.layout.simple list item 1, list);
48         //为 listview 绑定适配器
49         listView.setAdapter(adapter);
50      }
51
52
53      }
54    //解析 xml 文件
55  private List<Student> parserXMl()
56  {
57      //实例化一个 SAX 解析工厂
58      SAXParserFactory factory=SAXParserFactory.newInstance();
59      List<Student>students=null;
60      try {
```

```
61      //获取 xml 解析器
62      XMLReader reader=factory.newSAXParser().getXMLReader();
63      students=new ArrayList<Student>();
64      reader.setContentHandler(new StudentHandler(students));
65      //解析 Assets 下的 student.xml 文件
66      reader.parse(new InputSource(SaxXMLActivity.
        this.getResources().getAssets().open("student.xml")));
67      } catch (Exception e) {
68      // TODO: handle exception
69      }
70      return students;
71  }
72  }
```

（4）整个解析过程的关键就是使用 StudentHandler 来解析 xml 文本。解析过程会依次调用 startDocument()、startElement()、character()、endElement()和 endDocument()。需要注意的是，preTAG 这个变量是用来存储当前节点名称的，在 endElement()中要记得把它置为空，否则可能引起一些误判断。

```
01 package com.supermario.saxxml;                //声明包语句
02~08 行为引入相关类，这里不再列举，请阅读光盘内容。
//……
09 public class StudentHandler extends DefaultHandler {
10     private String preTAG;                    //用于存储 xml 节点的名称
11     private List<Student> ListStudent;
12     private Student stu;
13     //无参数实例化类
14     public StudentHandler() {
15         super();
16     }
17     //带参数实例化类
18     public StudentHandler(List<Student> listStudent) {
19         super();
20         ListStudent = listStudent;
21     }
22     //开始解析文档
23     public void startDocument() throws SAXException {
24         // TODO Auto-generated method stub
25     Log.i("------>", "文档开始");
26         super.startDocument();
27     }
28     //开始解析文档的元素
29     public void startElement(String uri, String localName, String qName,
30             Attributes attributes) throws SAXException {
31         Log.i("localName-------->", localName);
32         preTAG=localName;                   //将当前元素的名称保存到 preTAG
33         if ("student".equals(localName)) {
34             stu=new Student();              //实例化一个 student 类
35             //将 ID 信息保存到 stu 中
36             stu.setId(Long.parseLong(attributes.getValue(0)));
37
38         for (int i = 0; i < attributes.getLength(); i++) {
39             Log.i("attributes-------->",String.valueOf(stu.getId()));
40             }
41         }
42         //这句话记得要执行
```

```
43          super.startElement(uri, localName, qName, attributes);
44      }
45
46      public void endDocument() throws SAXException {
47
48          Log.i("------>", "文档结束");
49          super.endDocument();
50      }
51      public void endElement(String uri, String localName, String qName)
52              throws SAXException {
53          preTAG="";
54          if ("student".equals(localName)) {
55          ListStudent.add(stu);
56          Log.i("-------->", "一个元素解析完成");
57          }
58          super.endElement(uri, localName, qName);
59      }
60      //解析节点文本内容
61      public void characters(char[] ch, int start, int length)
62          throws SAXException {
63
64          String str;
65          //找出元素中的 name 节点
66         if ("name".equals(preTAG)) {
67             str=new String(ch,start,length);
68              stu.setName(str);
69              Log.i("name=", stu.getName());
70          //找出元素中的 speciality 节点
71          }else if (speciality.equals(preTAG)) {
72              str=new String(ch,start,length);
73              stu.setSpeciality(str);
74              Log.i("speciality=", stu.getSpeciality());
75          //找出元素中的 qq 节点
76          }else if ("qq".equals(preTAG)) {
77              str=new String(ch,start,length);
78              stu.setQQ(Long.parseLong((str)));
79              Log.i("QQ=", String.valueOf(stu.getQQ()));
80          }
81          super.characters(ch, start, length);
82      }
83      public List<Student> getListStudent() {
84          return ListStudent;
85      }
86
87      public void setListStudent(List<Student> listStudent) {
88          ListStudent = listStudent;
89      }
90  }
```

（5）最后我们还是来看看执行的结果，如图 13.3 所示。

13.2.3　PULL 解析

通过以上两种解析方式的分析，我们知道对往往内存比较稀缺的移动设备上运行的方



图 13.3　SAX 解析 XML

Android 系统来说，SAX 是一种比较合适的 XML 解析方式。但是 SAX 方式的特点是需要解析完整个文档才会返回，如果在一个 XML 文档中我们只需要前面的一部分数据，但是使用 SAX 方式还是会对整个文档进行解析，尽管 XML 文档中后面的大部分数据我们其实都不需要解析，因此这样实际上就浪费了处理资源。

Android 系统还提供了另一种 XML 解析方式，可以使你更好地处理这种情况，就是 PULL 方式解析 XML 数据。

PULL 解析器和 SAX 解析器虽有区别但也有相似性。SAX 解析器的工作方式是自动将事件推入注册的事件处理器进行处理，因此你不能控制事件的处理主动结束；而 PULL 解析器的工作方式为允许你的应用程序代码主动从解析器中获取事件，正因为是主动获取事件，因此可以在满足了需要的条件后不再获取事件，结束解析。这是它们主要的区别。

而它们的相似性在运行方式上，PULL 解析器也提供了类似 SAX 的事件（开始文档 START_DOCUMENT 和结束文档 END_DOCUMENT，开始元素 START_TAG 和结束元素 END_TAG，遇到元素内容 TEXT 等），但需要调用 next()方法提取它们（主动提取事件）。

Android 系统中和 PULL 方式相关的包为 org.xmlpull.v1，在这个包中提供了 PULL 解析器的工厂类 XmlPullParserFactory 和 PULL 解析器 XmlPullParser，XmlPullParserFactory 实例调用 newPullParser 方法创建 XmlPullParser 解析器实例，接着 XmlPullParser 实例就可以调用 getEventType()和 next()等方法依次主动提取事件，并根据提取的事件类型进行相应的逻辑处理。

下面我们仍然以一个例子来学习一下这种解析方式。

这里我们使用跟 13.2.2 小节相同的 xml 数据文件，类似的布局文件（只改了按钮的名称），以及同一个 student 类，用于存储 student 的信息。相同的代码部分这里不再赘述了，我们主要来看一下解析过程中的关键函数 private List<Student> parserXML()具体是如何实现的。

```
001  //解析 xml 文件
002 private List<Student> parserXMl()
003 {
004    //初始化一个 List<student>变量，用于存放所有 student 成员
005    List<Student> students=null;
006    //初始化一个 student 变量，用于存储每一个节点的信息
007    Student stu=null;
008    try{
```

```
009         //打开资源文件 student.xml
010         InputStream inputstream=PullXMLActivity.
            this.getResources().getAssets().open("student.xml");
011         //创建 XmlParser 有两种方式
012         //方式一：使用工厂类 XmlPullParserFactory
013         XmlPullParserFactory pullFactory=XmlPullParserFactory.
            newInstance();
014         XmlPullParser xmlPullParser=pullFactory.newPullParser();
015         //方式二：使用 Android 提供的实用工具类 android.util.Xml
016         //XmlPullParser xmlPullParser=Xml.newPullParser();
017         //设置输入字节流为 inputstream，并设置编码方式为 UTF-8
018         xmlPullParser.setInput(inputstream, "UTF-8");
019         //取得事件类型，用于开始解析时的判断
020         int eventType=xmlPullParser.getEventType();
021         //循环遍历整个文件直到解析完毕
022         while(eventType != XmlPullParser.END_DOCUMENT)
023         {
024             /*输出 log 显示事件类型
025             *START_DOCUMENT:0
026             *END_DOCUMENT:1
027             *START_TAG:2
028             *END_TAG:3
029             *TEXT:4
030             */
031             Log.e("guojs--->event",eventType+"");
032             //用于存储节点名称
033             String localName=null;
034             switch(eventType)
035             {
036             case XmlPullParser.START_DOCUMENT:
037                 //碰到文档开头则实例化 students 变量，并输出 log
038                 students=new ArrayList<Student>();
039                 Log.e("guojs","start document!");
040                 break;
041             case XmlPullParser.START_TAG:
042             {
043                 localName=xmlPullParser.getName();
044                 if ("student".equals(xmlPullParser.getName())) {
045                     stu=new Student();  //实例化一个 student 类
046                     //将 ID 信息保存到 stu 中
047                     stu.setId(Long.parseLong(xmlPullParser.
                    getAttributeValue(0)));
048                     Log.e("guojs",stu.getId()+"");
049                 }
050                 else if(stu != null)
051                 {
052                     //声明一个变量用于存储节点文本
053                     String currentData=null;
054                     if("name".equals(xmlPullParser.getName()))
055                     {
056                         /*注意这里 nextText()的使用：当前事件为 START_TAG，
057                          * 如果接下去是文本，就会返回当前的文本内容；如果下一个事
                         件是 END_TAG，
058                          * 就会返回空字符串；否则抛出一个异常
059                          */
060                         currentData=xmlPullParser.nextText();
061                         //存储 name 的信息
062                         stu.setName(currentData);
```

```
063                     }
064                     else if("speciality".equals
                       (xmlPullParser.getName()))
065                     {
066                         currentData=xmlPullParser.nextText();
067                         //存储专业信息
068                         stu.setSpeciality(currentData);
069                     }else if("qq".equals(xmlPullParser.getName()))
070                     {
071                         currentData=xmlPullParser.nextText();
072                         //存储 QQ 信息
073                         stu.setQQ(Long.parseLong(currentData));
074                     }
075                 }
076             }
077             break;
078             case XmlPullParser.END TAG:
079             {
080                 localName=xmlPullParser.getName();
081                 Log.e("guojs--end tag",localName);
082                 if("student".equals(localName) && stu != null)
083                 {
084                     //将 stu 添加进 students 数组列表中
085                     students.add(stu);
086                     //设置 stu 为空
087                     stu = null;
088                 }
089             }
090             break;
091             default:
092                 break;
093             }
094             //解析下一个事件
095             eventType=xmlPullParser.next();
096         }
097     }catch(Exception e)
098     {
099         e.printStackTrace();
100     }
101     return students;
102 }
```

通过上面的代码，我们发现，解析过程的关键就是判断事件的类型，然后在相应的事件中进行处理。不知大家发现没有，这里我们没有使用 TEXT 事件，为什么呢？其实 TEXT 事件已经在 START_TAG 中处理了，currentData=xmlPullParser.nextText()。在 START_TAG 中我们比较容易获得当前的 TAG 名称，然后进行一些判断，获取 TEXT 内容跟 TAG 的名称息息相关，所以我们这里不做专门针对 TEXT 事件的处理。

最后我们来看一下运行的效果图，如图 13.4 所示。

13.3　界 面 设 计

前面我们已经看过这个程序的效果图，现在我们将此效果图以代码的形式呈现出来，

第 13 章　Google 天气客户端



图 13.4　PULL 解析 XML

我们选择两种形式让用户输入，一种是用户从预先设置的城市中选择，一种是用户自己输入城市名称。用户单击对应的“确定”按钮就会触发相应的事件。根据用户的输入，进行相关查询，将结果显示到最下面的表格布局中。其中第一行显示的是今天的天气预报，接下去是往后 4 天的天气预报。此外，我们可以根据自己的喜好选择一张背景图片，如以下代码 006 行所示。

```
001 <?xml version="1.0" encoding="utf-8"?>
002 <LinearLayout xmlns:android="http: //schemas.android.
    com/apk/res/android"
003     android:orientation="vertical"
004     android:layout_width="fill_parent"
005     android:layout_height="fill_parent"
006     android:background="@drawable/bg">
007     <!-- 提示用户输入或选择城市名称 -->
008     <TextView
009         android:id="@+id/title"
010         android:layout width="wrap content"
011         android:layout height="wrap content"
012         android:text="@string/inputstr"
013         android:textStyle="bold"
014         android:textSize="16px"
015         android:layout_marginLeft="10px"
016         android:textColor="@color/black" />
017     <!-- 以表格形式绘制输入界面 -->
018     <TableLayout
019          android:id="@+id/TableLayout1"
020          android:layout height="wrap content"
021          android:layout width="fill parent">
022         <TableRow
023             android:id="@+id/TableRow01"
024             android:layout height="wrap content"
025             android:layout width="fill parent">
026             <!-- 请输入城市 -->
027            <TextView
028                android:id="@+id/city1Tv"
029                android:layout_width="wrap_content"
030                android:layout_height="wrap_content"
031                android:text="@string/msg"
032                android:textStyle="bold"
033                android:textSize="16px"
034                android:layout_marginLeft="10px"
```

```
035                  android:textColor="@color/black" />
036             <!-- 绘制 spinner 供用户选择城市 -->
037             <Spinner
038                 android:id="@+id/citySpinner"
039                 android:layout_height="wrap_content"
040                 android:layout_width="fill_parent"
041                 android:paddingLeft="10px"
042                 android:minWidth="200px" />
043             <!-- “确定”按钮 -->
044             <Button
045                 android:id="@+id/btn1"
046                 android:layout width="wrap content"
047                 android:layout height="wrap content"
048                 android:text="@string/OK"
049                 android:paddingLeft="10px" />
050         </TableRow>
051     </TableLayout>
052     <TableLayout
053         android:id="@+id/TableLayout2"
054         android:layout_height="wrap_content"
055         android:layout_width="fill_parent">
056         <TableRow
057             android:id="@+id/TableRow001"
058             android:layout_height="wrap_content"
059             android:layout_width="fill_parent">
060             <!-- 请输入城市 -->
061             <TextView
062                 android:id="@+id/cityTv2"
063                 android:layout width="wrap content"
064                 android:layout height="wrap content"
065                 android:text="@string/msg"
066                 android:textStyle="bold"
067                 android:textSize="16px"
068                 android:layout marginLeft="10px"
069                 android:textColor="@color/black" />
070             <!-- 城市输入框 -->
071             <EditText
072                 android:id="@+id/cityEt"
073                 android:layout height="wrap content"
074                 android:layout width="fill parent"
075                 android:paddingLeft="10px"
076                 android:minWidth="200px"></EditText>
077             <!-- 确定按钮 -->
078             <Button
079                 android:id="@+id/btn2"
080                 android:layout_width="wrap_content"
081                 android:layout_height="wrap_content"
082                 android:text="@string/OK"
083                 android:paddingLeft="10px" />
084         </TableRow>
085     </TableLayout>
086     <!-- 显示当前天气预报的城市 -->
087     <RelativeLayout
088         android:layout width="match parent"
089         android:layout height="wrap content" >
090         <TextView
091             android:id="@+id/currMsg"
092             android:layout height="wrap content"
093             android:layout_width="wrap_content"
094             android:textColor="#000000"
```

第 13 章　Google 天气客户端

```
095              android:text="当前城市是："
096             />
097         <TextView
098              android:id="@+id/currentCity"
099              android:text=""
100              android:layout_height="wrap_content"
101              android:layout_width="wrap_content"
102              android:layout_toRightOf="@+id/currMsg"
103              android:textColor="#336633"
104               />
105     </RelativeLayout>
106     <TableLayout
107         android:id="@+id/TableLayout3"
108         android:layout_width="fill_parent"
109         android:layout_height="wrap_content">
110         <!-- 今天天气  -->
111         <TableRow
112             android:id="@+id/TableRow02"
113             android:layout_width="wrap_content"
114             android:layout_height="wrap_content">
115             <com.guo.CityWeather.SingleWeatherInfoView
116                 android:id="@+id/weather_0"
117                 android:layout_width="wrap_content"
118                 android:layout_height="wrap_content"/>
119         </TableRow>
120             <!-- 明日天气 -->
121         <TableRow
122             android:id="@+id/TableRow03"
123             android:layout_width="wrap_content"
124             android:layout_height="wrap_content">
125             <com.guo.CityWeather.SingleWeatherInfoView
126                 android:id="@+id/weather_1"
127                 android:layout_width="wrap_content"
128                 android:layout_height="wrap_content"/>
129         </TableRow>
130             <!-- 后天天气 -->
131         <TableRow
132             android:id="@+id/TableRow04"
133             android:layout_width="wrap_content"
134             android:layout_height="wrap_content">
135             <com.guo.CityWeather.SingleWeatherInfoView
136                 android:id="@+id/weather_2"
137                 android:layout_width="wrap_content"
138                 android:layout_height="wrap_content"/>
139         </TableRow>
140             <!-- 大后天天气 -->
141         <TableRow
142             android:id="@+id/TableRow05"
143             android:layout_width="wrap_content"
144             android:layout_height="wrap_content">
145             <com.guo.CityWeather.SingleWeatherInfoView
146                 android:id="@+id/weather_3"
147                 android:layout_width="wrap_content"
148                 android:layout_height="wrap_content"/>
149         </TableRow>
150             <!-- 第四天天气 -->
151         <TableRow
152             android:id="@+id/TableRow06"
153             android:layout_width="wrap_content"
154             android:layout_height="wrap_content">
```

```
155             <com.guo.CityWeather.SingleWeatherInfoView
156                 android:id="@+id/weather 4"
157                 android:layout width="wrap content"
158                 android:layout height="wrap content"/>
159         </TableRow>
160     </TableLayout>
161
162 </LinearLayout>
```

其中，SingleWeatherInfoView 是我们自定义的用于显示天气预报信息的视图类，继承于 LinearLayout，包含两个元素，如下面代码所示。一个是 ImageView，用于显示天气状况图片，这个图片是从 Google 天气预报信息中提供的图片地址获得的；另一个是 TextView，用于显示天气详细状况。

```
01  package com.guo.CityWeather;                    //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容
//……
08  //新建一个视图继承 LinearLayout，用于显示天气预报信息
09  public class SingleWeatherInfoView extends LinearLayout
10  {
11      //用于显示天气状况图片
12      private ImageView  myWeatherImageView  = null;
13      //用于显示天气详细信息
14      private TextView   myTempTextView     = null;
15
16      public SingleWeatherInfoView(Context context)
17      {
18          super(context);
19      }
20      public SingleWeatherInfoView(Context context, AttributeSet attrs)
21      {
22          super(context, attrs);
23          //设置图像位置等信息
24          this.myWeatherImageView = new ImageView(context);
25          this.myWeatherImageView.setPadding(10, 5, 5, 5);
26          //设置文本颜色、字体大小
27          this.myTempTextView = new TextView(context);
28          this.myTempTextView.setTextColor(R.color.black);
29          this.myTempTextView.setTextSize(16);
30          //将 ImageView 元素添加到当前的 LinearLayout
31          this.addView(this.myWeatherImageView, new LinearLayout.
            LayoutParams(LayoutParams.WRAP_CONTENT, Layout
            Params.WRAP_CONTENT));
32          //将 TextView 元素添加到当前的 LinearLayout
33          this.addView(this.myTempTextView, new LinearLayout.
            LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.
            WRAP_CONTENT));
34      }
35      //设置文本内容
36      public void setWeatherString(String aWeatherString)
37      {
38          this.myTempTextView.setText(aWeatherString);
39      }
40      //设置图片
41      public void setWeatherIcon(Bitmap bm)
42      {
43          this.myWeatherImageView.setImageBitmap(bm);
44      }
```

```
45  }
```

以上就是所有界面实现的部分。

13.4 功 能 实 现

首先我们看一下需要解析的 XML 文件的内容形式。以 http://www.google.com.hk/ig/api?hl=zh_cn&weather=北京这个网页的内容为例。

```
01  <xml api reply version="1">
02      <weather module id="0" tab id="0" mobile row="0" mobile zipped="1"
        row="0" section="0">
03      <forecast information>
04          <city data="Beijing, Beijing"/>
05          <postal_code data="北京"/>
06          <latitude_e6 data=""/>
07          <longitude_e6 data=""/>
08          <forecast date data="2012-08-19"/>
09          <current date time data="2012-08-19 16:00:00 +0000"/>
10          <unit system data="SI"/>
11      </forecast information>
12          <current conditions>
13          <condition data="晴"/>
14          <temp_f data="82"/>
15          <temp_c data="28"/>
16          <humidity data="湿度： 48%"/>
17          <icon data="/ig/images/weather/sunny.gif"/>
18          <wind condition data="风向： 东北、风速：4 米/秒"/>
19      </current conditions>
20      <forecast_conditions>
21          <day_of_week data="周日"/>
22          <low data="18"/>
23          <high data="33"/>
24          <icon data="/ig/images/weather/mostly sunny.gif"/>
25          <condition data="以晴为主"/>
26      </forecast conditions>
27      <forecast conditions>
28          <day_of_week data="周一"/>
29          <low data="18"/>
30          <high data="31"/>
31          <icon data="/ig/images/weather/chance of storm.gif"/>
32          <condition data="可能有暴风雨"/>
33      </forecast conditions>
34      <forecast conditions>
35          <day_of_week data="周二"/>
36          <low data="13"/>
37          <high data="30"/>
38          <icon data="/ig/images/weather/sunny.gif"/>
39          <condition data="晴"/>
40      </forecast conditions>
41      <forecast conditions>
42          <day_of_week data="周三"/>
43          <low data="11"/>
44          <high data="32"/>
45          <icon data="/ig/images/weather/mostly_sunny.gif"/>
```

```
46          <condition data="晴间多云"/>
47      </forecast conditions>
48      </weather>
49  </xml_api_reply>
```

通过分析这些数据的组成我们发现，它主要由 3 部分组成，第一部分是天气预报的信息，主要包括城市信息、经纬度、时间等；第二部分是当前天气的信息，包含的内容比较详细，有天气状况、风向、温度、湿度、图标、风速等；第三部分为接下去四天的天气预报信息，包括星期几、最高气温、最低气温、天气状况、图标等信息。

根据需要，我们创建 3 个类来分别存储这些信息。

13.4.1　设置当前天气类

当前天气类 WeatherCurrentCondition.java，主要包括天气状况、摄氏温度、华氏温度、湿度、风向、图标网址、图标等信息。

```
001 package com.guo.CityWeather;
002
003 import android.graphics.Bitmap;
004
005 public class WeatherCurrentCondition
006 {
007
008     private String  condition;          //多云
009     private String  temp_celcius;       //摄氏温度
010     private String  temp fahrenheit;    //华氏温度
011     private String  humidity;           //湿度：58%
012     private String  wind_condition;     //风向
013     private String  icon;               //图标网址
014     private Bitmap bm;                  //图标
015
016     public WeatherCurrentCondition()
017     {
018
019     }
020     //得到 Condition（多云）
021     public String getCondition()
022     {
023         return condition;
024     }
025     //设置 Condition（多云）
026     public void setCondition(String condition)
027     {
028         this.condition = condition;
029     }
030     //得到摄氏温度
031     public String getTemp c()
032     {
033         return temp_celcius;
034     }
035     //得到华氏温度
036     public String getTemp f()
037     {
038         return temp_fahrenheit;
```

第 13 章　Google 天气客户端

```
039     }
040     //设置摄氏温度
041     public void setTemp celcius(String temp celcius)
042     {
043        this.temp_celcius = temp_celcius;
044     }
045     //设置华氏温度
046     public void setTemp fahrenheit(String temp fahrenheit)
047     {
048        this.temp fahrenheit = temp fahrenheit;
049     }
050     //得到湿度：58%
051     public String getHumidity()
052     {
053        return humidity;
054     }
055     //设置湿度：58%
056     public void setHumidity(String humidity)
057     {
058        this.humidity = humidity;
059     }
060     //得到风向指示
061     public String getWind_condition()
062     {
063        return wind_condition;
064     }
065     //设置风向指示
066     public void setWind condition(String wind condition)
067     {
068        this.wind condition = wind condition;
069     }
070     //得到图标地址
071     public String getIcon()
072     {
073        return icon;
074     }
075     //设置图标地址
076     public void setIcon(String icon)
077     {
078        this.icon = icon;
079     }
080     //设置图标
081     public void setBm(Bitmap bm)
082     {
083        this.bm = bm;
084     }
085     //得到图标
086     public Bitmap getBm()
087     {
088        return bm;
089     }
090     //得到一个封装打包的字符串，包括除 icno 外的所有东西
091     public String toString()
092     {
093        StringBuilder sb = new StringBuilder();
094        sb.append("实时天气: ").append(temp celcius).append(" °C");
095        sb.append(" ").append(temp_fahrenheit).append(" F");
096        sb.append(" ").append(condition);
097        sb.append(" ").append(humidity);
```

```
098         sb.append(" ").append(wind condition);
099         return sb.toString();
100     }
101 }
102
```

13.4.2　设置天气预报类

天气预报类 WeatherForecastCondition.java，用来存储未来几天的天气信息，主要包括星期、最低温度、最高温度、图标网址、图标等信息。

```
01  package com.guo.CityWeather;
02
03  import android.graphics.Bitmap;
04  public class WeatherForecastCondition {
05
06      private String day_of_week;    //星期
07      private String low;            //最低温度
08      private String high;           //最高温度
09      private String icon;           //图标网址
10      private String condition;      //天气状况
11      private Bitmap bm;              //图标
12      //不带参数初始化天气类
13      public WeatherForecastCondition()
14      {
15
16      }
17      //获取天气条件
18      public String getCondition()
19      {
20          return condition;
21      }
22
23      //设置天气预报
24      public void setCondition(String condition)
25      {
26          this.condition = condition;
27      }
28
29      //获取星期
30      public String getDay_of_week()
31      {
32          return day_of_week;
33      }
34
35      //设置星期
36      public void setDay of week(String day of week)
37      {
38          this.day of week = day of week;
39      }
40
41      //获取最低温度
42      public String getLow()
43      {
44          return low;
45      }
```

```
46
47      //设置最低温度
48      public void setLow(String low)
49      {
50          this.low = low;
51      }
52
53      //获取最高温度
54      public String getHigh()
55      {
56          return high;
57      }
58
59      //设置最高温度
60      public void setHigh(String high)
61      {
62          this.high = high;
63      }
64
65      //取得图标网址
66      public String getIcon()
67      {
68          return icon;
69      }
70
71      //设置图标网址
72      public void setIcon(String icon)
73      {
74          this.icon = icon;
75      }
76
77      //设置图标
78      public void setBm(Bitmap bm)
79      {
80          this.bm = bm;
81      }
82
83      //得到图标
84      public Bitmap getBm()
85      {
86          return bm;
87      }
88      public String toString()
89      {
90          StringBuilder sb = new StringBuilder();
91
92          sb.append(" ").append(day of week);
93          sb.append(" : ").append(high);
94          sb.append("/").append(low).append(" °C");
95          sb.append(" ").append(condition);
96          return sb.toString();
97      }
98  }
```

13.4.3　天气预报信息汇总

天气预报信息汇总类，封装了一个用于存储当前天气预报信息的成员变量 private

WeatherCurrentCondition myCurrentCondition 和一个用于存储未来几天信息的成员变量 private ArrayList<WeatherForecastCondition> myForecastConditions。

```
01  package com.guo.CityWeather;
02
03  import java.util.ArrayList;
04
05  public class WeatherSet
06  {
07      //实时天气信息
08      private WeatherCurrentCondition myCurrentCondition = null;
09      //预报的后四天的天气信息
10      private ArrayList<WeatherForecastCondition> myForecastConditions =
11      new ArrayList<WeatherForecastCondition>();
12      //实例化天气预报信息汇总类
13      public WeatherSet()
14      {
15
16      }
17
18      //得到实时天气信息的对象
19      public WeatherCurrentCondition getMyCurrentCondition()
20      {
21          return myCurrentCondition;
22      }
23
24      //设置实时天气信息的对象
25      public void setMyCurrentCondition(WeatherCurrentCondition
        myCurrentCondition)
26      {
27          this.myCurrentCondition = myCurrentCondition;
28      }
29
30      //得到预报天气
31      public ArrayList<WeatherForecastCondition> getMyForecast
        Conditions()
32      {
33          return myForecastConditions;
34      }
35
36      //得到最后一个预报天气
37      //这里我们每次添加一个数据都是在最后
38      //所以得到最后一个
39      public WeatherForecastCondition getLastForecastCondition()
40      {
41          return myForecastConditions.get(myForecastConditions.size()-1);
42      }
43  }
```

13.4.4　设置主界面

接下来要开始处理主界面的一些相关事件，如用户单击、内容显示等。如下面代码所示，在 onCreate 函数中执行初始化函数 init()，在 init()中首先为 spinner 绑定适配器，使得用户单击这个 spinner 的时候可以弹出城市选择菜单。接着为两个按钮分别绑定监听器，当

第 13 章　Google 天气客户端

用户单击 btn1 的时候，会根据用户选择的城市信息确定需要查询的 url 地址 url = new URL(ConstData.queryString + cityParamString)，接着通过函数 getCityWeather(url)取得相应的信息，并显示数据到主界面。当用户单击 btn2 按钮时，流程与 btn1 差不多，只不过 cityParamString 部分变成城市的名称。

这里要注意对这个城市名称进行一下处理，因为获取到的是中文，因此要考虑是否是 UTF8 编码，若不是则要转换成 UTF8 编码，否则就会出现找不到对应网址的错误。判断是否是汉字的方法很简单，依次获取字符串的每一个字符，对每个字符进行判断，大于 2 个字节的即是中文（这里不考虑其他语言的情况），然后通过 URLEncoder.encode(s_ch_one,"utf8")将其编码转换为 UTF8 编码。我们还定义了一个对话框，当没有获得指定网站的信息时，将会弹出“没有找到相关信息”的提示。

```
001 //用于显示城市信息
002 private String cityNow="";
003 //Google 天气预报的基准网址
004 private static String GOOGLE="http://www.google.com.hk";
005 private URL url;
006 final int DIALOG YES NO MESSAGE=1;
007 /** Called when the activity is first created. */
008 @Override
009 public void onCreate(Bundle savedInstanceState)
010 {
011     super.onCreate(savedInstanceState);
012     setContentView(R.layout.main);
013     //初始化界面
014     init();
015 }
016 //界面初始化
017 private void init()
018 {
019     Spinner city spr = (Spinner) findViewById(R.id.citySpinner);
020     //新建适配器，绑定城市数据
021    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
       android.R.layout.simple spinner item, ConstData.city);
022     //设置下拉菜单的布局
023     adapter.setDropDownViewResource(android.R.layout.simple
        spinner dropdown item);
024     //为 spinner 绑定适配器
025     city spr.setAdapter(adapter);
026 
027     Button submit = (Button) findViewById(R.id.btn1);
028     //为按钮绑定按键监听器
029     submit.setOnClickListener(new OnClickListener() {
030         @Override
031         public void onClick(View v)
032         {
033             // TODO Auto-generated method stub
034             Spinner spr = (Spinner) findViewById(R.id.citySpinner);
035             //取得选中 item 的 id 值
036             Long l = spr.getSelectedItemId();
037             //将 long 型转换为 int 型
```

```
038             int index = l.intValue();
039             //通过城市的 id 值取得经纬度信息
040             String cityParamString = ConstData.cityCode[index];
041             //取得当前选中的城市的名称
042             cityNow=(String) spr.getSelectedItem();
043             try
044             {
045                 //取得天气预报的 url 地址
046                 url = new URL(ConstData.queryString + cityParamString);
047                 new Thread(){
048                     public void run()
049                     {
050                         //获取天气信息
051                         getCityWeather(url);
052                     }
053                 }.start();
054             }
055             catch (Exception e)
056             {
057                 //如果出错则显示对话框提示用户
058                 showDialog1(DIALOG_YES_NO_MESSAGE);
059             }
060         }
061     });
062     Button submit_input = (Button) findViewById(R.id.btn2);
063     //为按钮绑定按键监听器
064     submit_input.setOnClickListener(new OnClickListener()
065        {
066         public void onClick(View v)
067         {
068             //输入框
069             EditText inputcity = (EditText) findViewById(R.id.cityEt);
070             //取得输入框的内容
071             final String tmp = inputcity.getText().toString();
072             //将城市名称存储到 cityNow 供接下去的显示用
073             cityNow=tmp;
074             new Thread(){
075                 public void run()
076                 {
077                     URL url;
078                     try {
079                         //取得天气预报的 url，注意这里中文字符串要转换成 UTF8，否
                        则会出错
080                         url = new URL(ConstData.queryString_intput + to_
                        Chanese(tmp));
081                         getCityWeather(url);
082                     } catch (MalformedURLException e) {
083                         //若出错则提示错误对话框
084                         showDialog1(DIALOG_YES_NO_MESSAGE);
085                     }
086                 }
087             }.start();
088         }
089        });
090 }
091 //显示对话框信息
092 void showDialog1(int id) {
093     CreateDialog(id).show();
094 }
```

```
095 //生成对话框
096 protected Dialog CreateDialog(int id) {
097     switch (id) {
098     case DIALOG_YES_NO_MESSAGE:
099         return new AlertDialog.Builder(this).setIcon(
100                 //设置标题：对不起；内容：没有找到相关信息
101                 R.drawable.alert dialog icon).setTitle(R.string.sorry)
102                 .setMessage(R.string.find nothing).setPositiveButton(
103                         R.string.conform,
104                         new DialogInterface.OnClickListener() {
105                             //设置按钮不作任何操作，直接关掉对话框
106                             public void onClick(DialogInterface dialog,
107                                     int whichButton) {
108                             }
109                         }).create();
110     }
111     return null;
112 }
113  //判断是否有汉字
114  public boolean vd(String str)
115  {
116      //取得字符串的字节数
117      byte[] bytes=str.getBytes();
118      //如果字节数大于 1 说明是汉字，否则不是
119      if (bytes.length>1)
120          return true;
121      else
122          return false;
123  }
124  //将汉字转成 UTF8 格式
125  public String to_Chianese(String str)
126  {
127      String s chin = "";
128      String s_ch_one;
129      for (int i=0;i<str.length();i++)
130      {
131          //依次截取每一个字符
132          s_ch_one=str.substring(i,i+1);
133          //检验每一字符
134          if (vd(s_ch_one))
135          {
136              try
137              {
138                  //如果是汉字则转换成 UTF8 的格式
139                  s_chin=s_chin+URLEncoder.encode(s_ch_one,"utf8");
140              }
141              catch (UnsupportedEncodingException e) {
142                  // TODO Auto-generated catch block
143                  e.printStackTrace();
144              }
145          }
146          else
147              s_chin=s_chin+s_ch_one;
148  }
149      //返回经过转换的字符串
150      return s chin;
151  }
```

13.4.5　ConstData.java 类

我们将所有的城市经纬度信息，以及查询网址的前缀等常量存放到 ConstData.java 这个类里面，方便进行统一管理和操作。

```
01  package com.guo.CityWeather;
02
03  public class ConstData
04  {
05      public static final String queryString="http: //www.google.com/ig/
        api?hl=zh-cn&weather=,,,";
06      public static final String queryString_intput="http://www.google.
        com/ig/api?hl=zh_cn&weather=";
07      public static final String [] cityCode ={
08              "39930000,116279998",          //北京
09              "31399999,121470001",          //上海
10              "39099998,117169998",          //天津
11              "29520000,106480003",          //重庆
12              "39669998,118150001",          //唐山
13              "38029998,114419998",          //石家庄
14              "38900001,121629997",          //大连
15              "45750000,126769996",          //哈尔滨
16              "20030000,110349998",          //海口
17              "43900001,125220001",          //长春
18              "28229999,112870002",          //长沙
19              "30670000,104019996",          //成都
20              "26079999,119279998",          //福州
21              "23129999,113319999",          //广州
22              "26579999,106720001",          //贵阳
23              "30229999,120169998",          //杭州
24              "31870000,117230003",          //合肥
25              "40819999,111680000",          //呼和浩特
26              "36680000,116980003",          //济南
27              "25020000,102680000",          //昆明
28              "29657589,91132050",           //拉萨
29              "36040000,103879997",          //兰州
30              "28600000,115919998",          //南昌
31              "32000000,118800003",          //南京
32              "22819999,108349998",          //南宁
33              "36069999,120330001",          //青岛
34              "22549999,114099998",          //深圳
35              "41770000,123430000",          //沈阳
36              "37779998,112550003",          //太原
37              "43779998,87620002",           //乌鲁木齐
38              "30620000,114129997",          //武汉
39              "34299999,108930000",          //西安
40              "36619998,101769996",          //西宁
41              "24479999,118080001",          //厦门
42              "34279998,117150001",          //徐州
43              "38479999,106220001",          //银川
44              "34720001,113650001"           //郑州
45      };
```

```
46      public static final String [] city ={
47          "北京",
48          "上海",
49          "天津",
50          "重庆",
51          "唐山",
52          "石家庄",
53          "大连",
54          "哈尔滨",
55          "海口",
56          "长春",
57          "长沙",
58          "成都",
59          "福州",
60          "广州",
61          "贵阳",
62          "杭州",
63          "合肥",
64          "呼和浩特",
65          "济南",
66          "昆明",
67          "拉萨",
68          "兰州",
69          "南昌",
70          "南京",
71          "南宁",
72          "青岛",
73          "深圳",
74          "沈阳",
75          "太原",
76          "乌鲁木齐",
77          "武汉",
78          "西安",
79          "西宁",
80          "厦门",
81          "徐州",
82          "银川",
83          "郑州"
84      };
85  }
```

13.4.6　程序的核心函数

接下来看看整个程序的核心函数 getCityWeather(url)，如以下代码 064 行所示。我们采用 SAX 解析的方式读取 xml 信息，13.2 节我们已经了解了 SAX 解析 xml 的原理，首先是实例化一个 SAX 工厂类 SAXParserFactory spf = SAXParserFactory.newInstance()，通过这个工厂类获得解析器 SAXParser sp = spf.newSAXParser()，再从这个解析器获取读取类 XMLReader xr = sp.getXMLReader()，最后为这个读取类设置内容处理类，并执行解析操作 xr.parse(is)。

通过解析最后得到天气预报信息 WeatherSet ws = gwh.getMyWeatherSet()。由于不是在主线程，因此我们要将信息更新到主界面的话需要借助消息机制。天气预报信息分别通过两个同名函数 updateWeatherInfoView，将信息绑定到 message，然后发送给 mHandler 进行处理。在 mHandler 中通过 msg.what 来区分是当前天气信息还是未来几天的天气信息，然后从消息绑定的数据中提取天气详细信息，更新到主界面中。

```
001 private Handler mHandler=new Handler(){
002     public void handleMessage(Message msg)
003     {
004         //当前天气信息
005         if(msg.what ==1)
006         {
007             //设置当前城市名称
008             ((TextView)findViewById(R.id.currentCity)).setText
                (cityNow);
009             int aResourceID=msg.arg1;
010             WeatherCurrentCondition aWCC=(WeatherCurrentCondition)
                msg.obj;
011             //设置图片
012             ((SingleWeatherInfoView) findViewById(aResourceID)). setWea
                therIcon(aWCC.getBm());
013             //设置天气预报详细信息
014             ((SingleWeatherInfoView) findViewById(aResourceID)). setWea
                therString(aWCC.toString());
015         }
016         //天气预报信息
017         else if(msg.what == 2)
018         {
019             int aResourceID=msg.arg1;
020             WeatherForecastCondition aWCC=(WeatherForecastCondition)
                msg.obj;
021             //设置图片
022             ((SingleWeatherInfoView) findViewById(aResourceID)). setWea
                therIcon(aWCC.getBm());
023             //设置天气预报详细信息
024             ((SingleWeatherInfoView)  findViewById(aResourceID)).
                setWeatherString(aWCC.toString());
025         }
026     }
027 };
028 //更新显示实时天气信息
029 private void updateWeatherInfoView(int aResourceID, WeatherCurrent
    Condition aWCC) throws MalformedURLException
030 {
031     //通过 url 地址获取位图信息
032     URL imgURL = new URL(GOOGLE + aWCC.getIcon());
033     Bitmap bm=getBm(imgURL);
034     //将位图存储到对应的类中
035     aWCC.setBm(bm);
036     Message msg=new Message();
037     msg.what=1;
038     //将类绑定到消息中
039     msg.obj=aWCC;
040     //将需要更新的界面元素的 id 绑定到消息中
041     msg.arg1=aResourceID;
042     mHandler.sendMessage(msg);
043 }
```

```
044 //更新显示天气预报
045 private void updateWeatherInfoView(int aResourceID, WeatherForecast
    Condition aWFC) throws MalformedURLException
046 {
047     //通过 url 地址获取位图信息
048     URL imgURL = new URL(GOOGLE + aWFC.getIcon());
049     Bitmap bm=getBm(imgURL);
050     //将位图存储到对应的类中
051     aWFC.setBm(bm);
052     Message msg=new Message();
053     msg.what=2;
054     //将类绑定到消息中
055     msg.obj=aWFC;
056     //将需要更新的界面元素的 id 绑定到消息中
057     msg.arg1=aResourceID;
058     mHandler.sendMessage(msg);
059 }
060
061 //获取天气信息
062 //通过网络获取数据
063 //传递给 XMLReader 解析
064 public void getCityWeather(URL url)
065 {
066     try
067     {
068         //新建一个 SAX 工厂类
069         SAXParserFactory spf = SAXParserFactory.newInstance();
070         //实例化 SAX 解析器
071         SAXParser sp = spf.newSAXParser();
072         //设置 SAX 读取类
073         XMLReader xr = sp.getXMLReader();
074         //设置解析器对应的处理类
075         GoogleWeatherHandler gwh = new GoogleWeatherHandler();
076         xr.setContentHandler(gwh);
077         //获取网页的数据流
078         InputStreamReader isr = new InputStreamReader(url.openStream(),
            "GBK");
079         //将数据流包装成 InputSource
080         InputSource is = new InputSource(isr);
081         //解析数据流
082         xr.parse(is);
083         //获得天气汇总信息
084         WeatherSet ws = gwh.getMyWeatherSet();
085         //通过天气汇总信息取得当前天气信息
086         updateWeatherInfoView(R.id.weather_0, ws.getMyCurrent
            Condition());
087         //通过天气汇总信息分别取得接下去四天的天气信息
088         updateWeatherInfoView(R.id.weather_1, ws.getMyForecastCondi
            tions().get(0));
089         updateWeatherInfoView(R.id.weather_2, ws.getMyForecastCondi
            tions().get(1));
090         updateWeatherInfoView(R.id.weather_3, ws.getMyForecastCondi
            tions().get(2));
091         updateWeatherInfoView(R.id.weather_4, ws.getMyForecastCondi
            tions().get(3));
092     }
093     catch (Exception e)
094     {
```

```
095         Log.e("CityWeather", e.toString());
096     }
097 }
098 //通过 url 地址取得天气状况图像
099 public Bitmap getBm(URL aURL)
100 {
101     URLConnection conn;
102     Bitmap bm=null;
103     try {
104         //打开 url 链接
105         conn = aURL.openConnection();
106         conn.connect();
107         //将数据流保存到 is
108         InputStream is = conn.getInputStream();
109         BufferedInputStream bis = new BufferedInputStream(is);
110         //用位图工厂解码数据流，将数据流转换成位图
111         bm = BitmapFactory.decodeStream(bis);
112         bis.close();
113         is.close();
114     } catch (IOException e) {
115         // TODO Auto-generated catch block
116         e.printStackTrace();
117     }
118     //返回位图
119     return bm;
120 }
```

13.4.7　存储天气信息

最后我们来看看这个 xml 内容处理类 GoogleWeatherHandler 具体是如何实现的。首先，新建一个继承 DefaultHandler 的类 googleWeatherHandler，在这个类里面设置了一个 private WeatherSet myWeatherSet 变量，用于存储天气信息；变量 private Boolean is_Current_Conditions 和 private boolean is_Forecast_Conditions 分别用于作为当前天气和天气预报的标识标量。因为我们这次解析的都是一些节点的属性，因此整个解析过程需要处理的部分都集中在 startElement()和 endElement()函数中。

在 startElement()中，如果遇到 CURRENT_CONDITIONS 或者 FORECAST_CONDITIONS，则表明开始解析当前天气或者未来天气，因此当遇到这两个标签的时候，需要新建一个相应的类用于存储信息并将标志位改为 true，以便在解析这两种类型的信息的相同部分（如 icon）时能够区分开来。然后在 endElement 中，如果遇到 CURRENT_CONDITIONS 或者 FORECAST_CONDITIONS，则表明对应的元素已经解析完成，需要将标志位改为 false。

```
001 package com.guo.CityWeather;                //声明包语句
002~006 行为引入相关类，这里不再列举，请阅读光盘内容
//……
007 public class GoogleWeatherHandler extends DefaultHandler
008 {
009     //天气信息
010     private WeatherSet      myWeatherSet           = null;
011
012     //实时天气信息
013     private boolean        is_Current_Conditions   = false;
```

```
014     //预报天气信息
015     private boolean        is Forecast Conditions  = false;
016
017     private final String   CURRENT CONDITIONS     = "current condi
        tions";
018     private final String   FORECAST CONDITIONS     = "forecast condi
        tions";
019
020     //不带参数实例化类
021     public GoogleWeatherHandler()
022     {
023
024     }
025     //返回天气信息对象
026     public WeatherSet getMyWeatherSet()
027     {
028         return myWeatherSet;
029     }
030     //文档结尾
031     @Override
032     public void endDocument() throws SAXException
033     {
034         // TODO Auto-generated method stub
035         super.endDocument();
036     }
037     //元素结尾
038     @Override
039     public void endElement(String uri, String localName, String name)
        throws SAXException
040     {
041         //如果遇到当前天气信息标签，则将相应标志位置为 false
042         if (localName.equals(CURRENT CONDITIONS))
043         {
044             this.is Current Conditions = false;
045         }
046         //如果遇到天气预报信息标签，则将相应标志位置为 false
047         else if (localName.equals(FORECAST CONDITIONS))
048         {
049             this.is Forecast Conditions = false;
050         }
051     }
052
053     //开始解析文档
054     @Override
055     public void startDocument() throws SAXException
056     {
057         this.myWeatherSet = new WeatherSet();
058     }
059
060     //开始解析元素
061     @Override
062     public void startElement(String uri, String localName, String name,
        Attributes attributes) throws SAXException
063     {
064         if (localName.equals(CURRENT_CONDITIONS))
```

```
065         {   //实时天气
066             this.myWeatherSet.setMyCurrentCondition(new WeatherCurrent
                Condition());
067             this.is_Current_Conditions = true;
068         }
069         else if (localName.equals(FORECAST_CONDITIONS))
070         {   //预报天气
071             this.myWeatherSet.getMyForecastConditions().add(newWeather
                ForecastCondition());
072             this.is_Forecast_Conditions = true;
073         }
074         else
075         {
076             //获取属性 data 的值
077             String dataAttribute = attributes.getValue("data");
078             //如果是 icon，则判断是当前天气的 icon 还是天气预报的 icon
079             if (localName.equals("icon"))
080             {
081                 if (this.is_Current_Conditions)
082                 {
083                      this.myWeatherSet.getMyCurrentCondition().setIcon
                         (dataAttribute);
084                 }
085                 else if (this.is_Forecast_Conditions)
086                 {
087                      this.myWeatherSet.getLastForecastCondition().set
                          Icon(dataAttribute);
088                 }
089             }
090             //如果是 condition，则判断是当前天气的 condition 还是天气预报的
                  condition
091             else if (localName.equals("condition"))
092             {
093                 if (this.is_Current_Conditions)
094                 {
095                         this.myWeatherSet.getMyCurrentCondition().set
                            Condition(dataAttribute);
096                 }
097                 else if (this.is_Forecast_Conditions)
098                 {
099                         this.myWeatherSet.getLastForecastCondition().set
                            Condition(dataAttribute);
100                 }
101             }
102             else if (localName.equals("temp_c"))
103             {
104                      this.myWeatherSet.getMyCurrentCondition().setTemp
                         celcius(dataAttribute);
105             }
106             else if (localName.equals("temp_f"))
107             {
108                       this.myWeatherSet.getMyCurrentCondition().
                          setTemp_fahrenheit(dataAttribute);
109             }
```

```
110             else if (localName.equals("humidity"))
111             {
112                     this.myWeatherSet.getMyCurrentCondition().
                        setHumidity(dataAttribute);
113             }
114             else if (localName.equals("wind condition"))
115             {
116                     this.myWeatherSet.getMyCurrentCondition().
                        setWind condition(dataAttribute);
117             }// Tags is forecast conditions
118             else if (localName.equals("day of week"))
119             {
120                     this.myWeatherSet.getLastForecastCondition().
                        setDay_of_week(dataAttribute);
121             }
122             else if (localName.equals("low"))
123             {
124                     this.myWeatherSet.getLastForecastCondition().
                        setLow(dataAttribute);
125             }
126             else if (localName.equals("high"))
127             {
128                     this.myWeatherSet.getLastForecastCondition().
                        setHigh(dataAttribute);
129             }
130         }
131     }
132     //如果遇到元素节点文本
133     @Override
134     public void characters(char ch[], int start, int length)
135     {
136     }
137 }
```

由于我们程序需要联网，因此我们还要在 AndroidManifest.xml 中增加使用 internet 的权限：

```
<uses-permission android:name="android.permission.INTERNET" />
```

到此，整个程序完成了，我们可以运行程序测试一下效果。如图 13.5 所示，我们从预设值的城市列表中选择一个城市——厦门，单击“确定”按钮，即可显示厦门的天气状况；手动输入城市名称——深圳，单击“确定”按钮，同样也显示了深圳的天气状况。

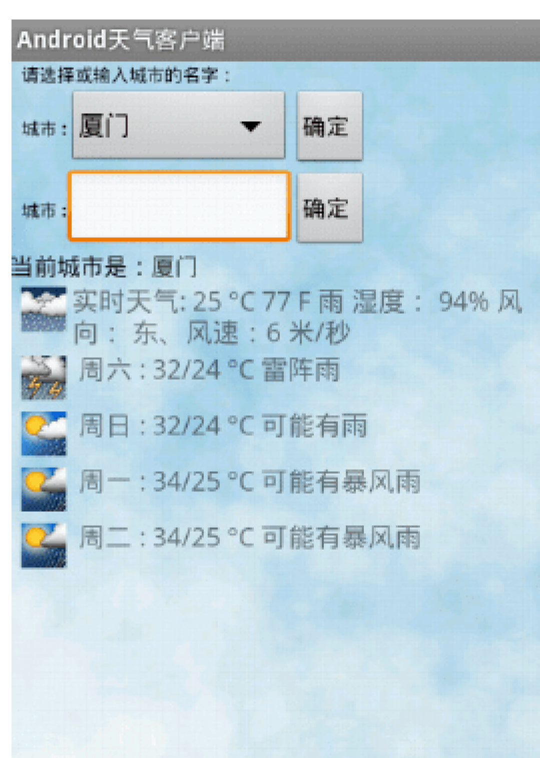


图 13.5　从列表中选择城市查询



图 13.6　手动输入城市名称查询

13.5　知 识 拓 展

网上有不少 Android 天气预报的 APK，大部分 APK 都附带有一个 widget，使用 widget 可以很方便地在桌面上显示天气信息。其实我们这个程序也可以改进一下，增加一个 widget 功能，这些就留给读者去实现吧。下面我们以一个小小的示例来讲解一下如何实现 widget。写一个 widget 的大致流程是：

- AppWidgetProvider 的实现；
- widget 外观布局定义文件；
- 新增 widget 时的配置 Activity 的实现（可选）；
- widget 参数配置文件；
- 修改 AndroidManifest.xml。

（1）我们先新建一个工程 MyWidget，项目结构如图 13.7 所示。

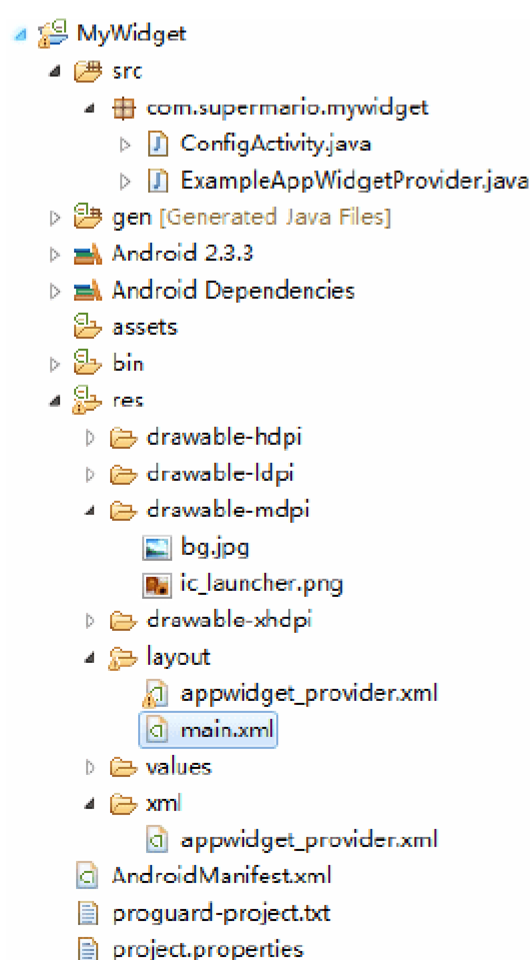


图 13.7　MyWidget 项目结构

（2）AppWidgetProvider 的实现。代码如下，在这里由于我们只是实现一个框架，不做具体的功能处理，因此我们只简单地在函数中输出一个 log。

```
01  package com.supermario.mywidget;                        //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
08  public class ExampleAppWidgetProvider extends AppWidgetProvider{
09      private String TAG="MyWidget";
10      //widget 被删除时调用
11      @Override
12      public void onDeleted(Context context, int[] appWidgetIds) {
13          // TODO Auto-generated method stub
14          super.onDeleted(context, appWidgetIds);
15          Log.i(TAG,"onDeleted");
16      }
```

```
17      //当最后一个 widget 实例被删除时调用
18      @Override
19      public void onDisabled(Context context) {
20          // TODO Auto-generated method stub
21          super.onDisabled(context);
22          Log.i(TAG,"onDisabled");
23      }
24      //当 widget 被创建时调用
25      @Override
26      public void onEnabled(Context context) {
27          // TODO Auto-generated method stub
28          super.onEnabled(context);
29          Log.i(TAG,"onEnabled");
30      }
31      //主要用于调度该 ExampleAppWidgetProvider 类中的其他方法
32      @Override
33      public void onReceive(Context context, Intent intent) {
34          // TODO Auto-generated method stub
35          super.onReceive(context, intent);
36          Log.i(TAG,"onReceive");
37      }
38      //当需要提供 RemoteViews 时调用
39      @Override
40      public void onUpdate(Context context, AppWidgetManager appWidget
        Manager, int[] appWidgetIds) {
41          // TODO Auto-generated method stub
42          super.onUpdate(context, appWidgetManager, appWidgetIds);
43          Log.i(TAG,"onUpdate");
44      }
45  }
```

（3）widget 外观布局定义文件。这个就是在桌面上的 widget 的样子，我们用的是一张图片@drawable/bg。

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <ImageView xmlns:android="http://schemas.android.com/apk/res/android"
3      android:id="@+id/img"
4      android:layout width="wrap content"
5      android:layout height="wrap content"
6      android:src="@drawable/bg"
7      android:clickable="true"/>
```

（4）新增 widget 时的配置 Activity 的实现（可选）。Android 平台为 widget 提供了一个配置界面的功能，我们可以自定义一个 Activity，在 widget 参数配置文件中配置好相关参数后，此 Activity 会在用户新增 widget 时自动调用。一般来说，这个配置界面的作用是用户新建 widget 时，让用户配置 widget 的一些属性，如颜色、大小等等。但是在我们的这个示例程序中，我们用它来当作创建便签的地方。

不过本节只是先实现一个原型程序，所以暂时不作处理，我们只是新建一个 Activity 即可。新建名为 ConfigActivity 的 Activity，重写 onCreate 方法。在 OnCreate 方法中，由于这个 Activity 是由系统在新增 widget 时自动调用的，所以我们可以用 getIntent 获取到传入的 widgetId。可以判断其是否是一个有效的 widgetId，最后我们必须返回一个 RESULT_OK 的 Intent，并结束当前 Activity，系统才会认为配置成功，在桌面上放置这个

widget。如果返回 RESULT_CANCELED，系统会认为配置失败，终止 widget 的创建过程。

```
01  package com.supermario.mywidget;                    //声明包语句
02~06 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
07  //该 Activity 在系统新增 widget 时被调用
08  public class ConfigActivity extends Activity{
09      private int mAppWidgetId;
10      @Override
11      protected void onCreate(Bundle savedInstanceState) {
12          // TODO Auto-generated method stub
13          super.onCreate(savedInstanceState);
14          //设置当前界面 layout 为 main.xml
15          setContentView(R.layout.main);
16          Log.i("ConfigActivity","onCreate!");
17          //取得启动这个 Activity 的 Intent
18          Intent intent = getIntent();
19          //取得该 intent 的扩展数据
20          Bundle extras = intent.getExtras();
21          //得到 widget 传过来的 id，每一个 widget 都有一个不相同的 id
22          if (extras != null) {
23              mAppWidgetId = extras.getInt(AppWidgetManager. EXTRA
                APPWIDGET_ID,
24                      AppWidgetManager.INVALID APPWIDGET ID);
25          }
26          Log.i("ConfigActivity",mAppWidgetId+"");
27          //如果没有传递 AppWidgetId，则直接结束
28          if (mAppWidgetId == AppWidgetManager.INVALID_APPWIDGET_ID) {
29              finish();
30          }
31          //最后我们必须返回一个 RESULT_OK 的 Intent，并结束当前 Activity，
32          //系统才会认为配置成功，在桌面上放置这个 widget{
33          Intent resultValue = new Intent();
34          resultValue.putExtra(AppWidgetManager.EXTRA APPWIDGET ID,
35                  mAppWidgetId);
36
37          setResult(RESULT_OK, resultValue);
38          finish();
39      }
40  }
```

（5）widget 参数配置文件。我们需要编写一个 widget 参数配置文件，将布局文件、配置 Activity 关联起来。我们在 res 下新建目录 xml，在 xml 目录下新增文件 appwidget_provider.xml，编写如下：

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <appwidget-provider xmlns:android="http://schemas.android.com
    /apk/res/android"
3       android:minWidth="72dp"
4       android:minHeight="72dp"
5       android:updatePeriodMillis="86400000"
6       android:initialLayout="@layout/appwidget_provider"
7       android:configure="com.supermario.mywidget.ConfigActivity"
```

```
8       >
9   </appwidget-provider>
```

（6）修改 AndroidManifest.xml。为了运行 widget，我们还需要修改一下 AndroidManifest.xml 来声明我们的 widget。声明一个 receiver，过滤 android.appwidget.action.APPWIDGET_UPDATE，并且用 metadata 关联到我们自己编写的 appWidgetProvider 实现。声明一个 Activity 关联到我们的配置类 MyNoteConf，过滤 android.appwidget.action.APPWIDGET_CONFIGURE。最后修改一下应用图标，此图标会出现在系统的新增 widget 列表中。

编写好的 AndroidManifest.xml 代码如下：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
03      package="com.supermario.mywidget"
04      android:versionCode="1"
05      android:versionName="1.0" >
06      <uses-sdk android:minSdkVersion="10" />
07      <application
08          android:icon="@drawable/ic_launcher"
09          android:label="@string/app_name" >
10          <receiver android:name=".ExampleAppWidgetProvider">
11              <intent-filter>
12                  <action android:name="android.appwidget.
                    action.APPWIDGET_UPDATE" />
13              </intent-filter>
14              <meta-data android:name="android.appwidget.provider"
15                      android:resource="@xml/appwidget_provider" />
16          </receiver>
17          <activity android:name=".ConfigActivity">
18              <intent-filter>
19                  <action android:name="android.appwidget.action.
                    APPWIDGET_CONFIGURE"/>
20              </intent-filter>
21          </activity>
22      </application>
23  </manifest>
```

（7）至此原型程序全部开发完成，运行一下看看效果吧，如图 13.8 所示。在桌面上长按，可以选择我们刚刚写的原型 widget “MyWidget” 了。选择后出现我们定义的配置界面 ConfigActivity，但是由于我们在 onCreate 中 finish 了，所以是一闪而过的。之后 MyWidget 就出现在桌面上了，我们可以随便拖动它，或者把它丢进垃圾箱，观察一下日志输出。



图 13.8　MyWidget 桌面效果图

13.6　本 章 小 结

本章主要讲述了 Android 天气预报的实现，顺便也给大家介绍了 Android 3 种 XML 解析方式的原理和实现，在本章最后还给大家介绍了 Android Widget 的简单实现。通过本章的学习，我们要熟练使用 Android 这 3 种 XML 解析方式，因为 XML 解析在编写 Android APK 的时候随时可能用到，熟练使用可以让我们在编写其他程序的时候游刃有余。最后，希望有心的读者可以改进一下这个 APK，加入 widget 实现，使这个程序更美观实用。

第 14 章　RSS 新闻阅读器

为什么使用 RSS？信息传播工具有多种多样，包括可以免费收听的无线电广播、公共和有线电视、印刷媒体，甚至包括 Internet 这样颠覆性的技术，以及庞大的 Web 站点和电子邮件订阅。虽然选择很多，但是这些工具都存在一个问题：很难在庞杂的海量数据中精确查找到真正感兴趣的信息和价值。幸运的是，RSS 可以帮助我们解决这个问题。

14.1　功 能 分 析

RSS（Really Simple Syndication）是一种内容发布者用来发布信息的 XML 数据格式，这些信息经过了分类并适合人机阅读。RSS 提要通常使用诸如新闻阅读器这种人类可读的友好格式进行处理并显示给用户，本章要设计的应用程序就是这样一种新闻阅读器。RSS 提要同样可以供计算机使用，从而生成后续的、聚合的信息源。RSS 的格式是 XML 数据，这表示数据本身就包含描述性元素，也就是说它是自包含的。随着行业的逐步规范化，XML 结构在过去几年也经历了一些变化。最新的版本也是应用最广的版本是 2.0。RSS 2.0 是一种相对简单的 XML 结构，很容易由计算机程序解析。

我们以新浪的 RSS 源为例，来分析一下 RSS 2.0 的结构，如图 14.1 所示。

```
<rss version="2.0">
    <channel>
        <title>
            <![CDATA[真情时刻-新浪新闻]]>
        </title>
        <image>
            <title>
                <![CDATA[新闻中心-社会新闻]]>
            </title>
            <link>http://news.sina.com.cn/society/index.shtml</link>
            <url>http://www.sinaimg.cn/home/deco/2009/0330/logo_home_news.gif</url>
        </image>
        <description>
            <![CDATA[社会新闻-真情时刻]]>
        </description>
        <link>http://news.sina.com.cn/society/index.shtml</link>
        <language>zh-cn</language>
        <generator>WWW.SINA.COM.CN</generator>
        <ttl>5</ttl>
        <copyright>
            <![CDATA[Copyright 1996 - 2012 SINA Inc. All Rights Reserved]]>
        </copyright>
        <pubDate>Sun, 4 Nov 2012 01:30:16 GMT</pubDate>
        <category>
            <![CDATA[]]>
        </category>
        <item>
            <title>
                <![CDATA[女子患抑郁症半夜爬上34楼护栏]]>
            </title>
```

图 14.1　新浪 RSS 文本截图

从上面可以看出，整个文本文档以 rss 为根节点，包含一个 channel，整个 xml 文本的节点结构如下所示：

```
01  <rss>
02  <channel>
03  <title />
04  <image>
05  <title />
```

```
06  <link />
07  <url />
08  </image>
09  <description />
10  <link />
11  <language />
12  <ttl />
13  <copyright />
14  <pubDate>
15  <item />
16  </channel>
17  </rss>
```

其中每个 item 包含的元素如下所示：

```
1   <item>
2   <title />
3   <link />
4   <author />
5   <guid />
6   <category />
7   <pubDate />
8   <comments />
9   </item>
```

通过每个节点的标签我们就可以猜测出节点包含的信息，本章程序需要完成的主要工作就是解析这种格式的 XML 文档，按照程序的需求组织和显示相应节点的信息。

14.2　登录过程实现

登录画面我们设置了显示一张图片和一个登录按钮，当单击按钮时画面透明度不断提高，最终完全消失，与此同时，关闭当前界面，并启动 RSS 源选择界面。登录画面如图 14.2 所示。



图 14.2　RSS 登录画面

14.2.1　界面设置

界面实现代码如下所示，以 LinearLayout 为根节点，从上到下依次为登录画面、登录文本、登录按钮，这里的登录文本一开始设置为不可见，如代码 23 行所示。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.   android.com
    /apk/res/android"
03     android:orientation="vertical"
04     android:layout_width="fill_parent"
05     android:layout_height="fill_parent"
06     >
07     <!-- 登录图片 -->
08     <ImageView
09         android:id="@+id/login_rss"
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:src="@drawable/rss"
13         android:layout_gravity="center_horizontal"
14         android:layout_marginTop="30dp"
15         />
16     <!-- 登录文本 -->
17     <TextView
18         android:id="@+id/logining"
19         android:layout_width="fill_parent"
20         android:layout_height="wrap_content"
21         android:gravity="center_horizontal"
22         android:text="@string/logining"
23         android:visibility="invisible"
24         android:layout_marginTop="20dp"
25         />
26     <!-- 登录按钮 -->
27     <Button
28         android:id="@+id/login_in"
29         android:layout_width="200dp"
30         android:layout_height="wrap_content"
31         android:layout_gravity="center_horizontal"
32         android:text="@string/login_in"
33         android:layout_marginTop="10dp"
34         />
35  </LinearLayout>
```

14.2.2　新建 LoginActivity.java

我们在包 com.rss.activity 下新建 LoginActivity.java，代码如下所示。在 onCreate 中初始化界面，为登陆按键绑定监听器，图片初始的透明度值为 255，表示完全不透明。当单击登录按钮时，将开启线程，并每隔 100ms 更新一次图片的透明度。如代码 76~91 行所示，图片每次透明度减少 25，当透明度低于 25 时，将图片设置为完全透明，此时停止更新界面，启动 RSS 频道选择界面 SelectChannel，并关闭当前界面。

```
01  package com.rss.activity;                //声明包语句
02~13 行为引入相关类，这里不再列举，请阅读光盘内容
```

```
//
……
14  public class LoginActivity extends Activity {
15      //登录按钮
16      private Button loginButton;
17      //登录文本
18      private TextView loginText;
19      //登录图片
20      private ImageView imagView;
21      //图片透明度
22      private int i_alpha = 255;
23      private Handler mHandler = new Handler();
24      boolean isShow = false;
25      private Thread thread;
26      private Intent intent;
27      @Override
28      protected void onCreate(Bundle savedInstanceState) {
29          // TODO Auto-generated method stub
30          super.onCreate(savedInstanceState);
31          setContentView(R.layout.login);
32          //初始化界面元素
33          loginButton = (Button)findViewById(R.id.login_in);
34          loginText = (TextView)findViewById(R.id.logining);
35          imagView = (ImageView)findViewById(R.id.login_rss);
36          //初始化图片分辨率
37          imagView.setAlpha(i alpha);
38          isShow = true;
39          //更改图片分辨率
40          mHandler = new Handler() {
41              @Override
42              public void handleMessage(Message msg) {
43                  super.handleMessage(msg);
44                  imagView.setAlpha(i_alpha);
45              }
46          };
47
48          //开启线程每隔 100ms 更新一次图片透明度
49          thread = new Thread(new Runnable() {
50              public void run() {
51                  while(isShow) {
52                      try {
53                          Thread.sleep(100);
54                          updateAlpha();
55                      }catch (InterruptedException  e) {
56                          e.printStackTrace();
57                      }
58                  }
59              }
60
61          });
62          //登录按键监听器
63          loginButton.setOnClickListener(new OnClickListener() {
64              public void onClick(View v) {
65                  intent = new Intent();
66                  intent.setClass(LoginActivity.this, SelectChannel.
                    class);
67                  //启动线程
68                  thread.start();
69                  //设置登录按钮不可见
```

```
70                  loginButton.setVisibility(View.INVISIBLE);
71                  //设置登录文本显示
72                  loginText.setVisibility(View.VISIBLE);
73              }
74          });
75      }
76      //更新图片透明度
77      protected void updateAlpha() {
78          //每次减 25
79          if((i_alpha-25) >= 0) {
80              i_alpha = i_alpha - 25;
81
82          }else {
83              //当透明度低于 25，关闭当前界面，启动新界面
84              i_alpha = 0;
85              isShow = false;
86              startActivity(intent);
87              LoginActivity.this.finish();
88          }
89          //传递消息
90          mHandler.sendMessage(mHandler.obtainMessage());
91      }
92 }
```

14.3　RSS 源的设置

14.3.1　RSS 源选择界面设计

登录界面之后是 RSS 源的显示界面，如图 14.3 所示，在该界面我们需要完成 RSS 源的添加、删除和选择功能。代码如下所示，在包 com.rss.activity 下新建 SelectChannel.java，新建类 SelectChannel 继承于 ListActivity。在 onCreate()函数中，初始化 RSS 源数据库类 ChannelDataHelper，关于这个类，接下去会进一步分析，这里我们只需要知道这个类会实现对 RSS 源的一系列数据库操作。函数 GetChannelList()用于获取数据库中所有的元素，若当前数据库中没有数据，则提示“尚未添加任何频道”，否则为当前界面设置适配器 mAdapter。



图 14.3　RSS 源选择界面

mAdapter 变量是一个 ChannelAdapter 的实例，ChannelAdapter 继承于 BaseAdapter，我们着重分析一下该类中 getView()函数的实现。如代码 069~124 行所示，首先初始化界面元素类 ViewHolder，膨胀出元素界面 R.layout.item，接着初始化 item 中的界面元素，为“删除”按钮和 textLayout 绑定监听器。

当单击 textLayout 按钮时，将当前单击元素对应的 RSS 源 url 地址传递给界面 ActivityMain，并启动界面 ActivityMain。当用户单击“删除”按钮时，将移除数据库中对应的数据，并更新当前界面的显示。代码 133~141 行添加 menu 菜单，一个为“添加 RSS 源”，另一个为退出程序。代码 144~159 行为菜单按键绑定监听器，当单击“添加 RSS”源时，将启动添加界面 AddRss，并关闭当前界面，当单击“退出”按钮时，将直接关闭当前界面退出程序。

```
001 package com.rss.activity;                //声明包语句
002~022 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
023 //RSS 源选择界面
024 public class SelectChannel extends ListActivity{
025     //menu 菜单的 id
026     private static final int MENU_ADD = Menu.FIRST;
027     private static final int MENU_QUIT = MENU_ADD + 1;
028     //RSS 源适配器
029     ChannelAdapter mAdapter;
030     //RSS 源数据库类
031     ChannelDataHelper mChannel;
032     //RSS 源数组
033     List<String> channelList=new ArrayList<String>();
034     @Override
035     public void onCreate(Bundle savedInstanceState) {
036         super.onCreate(savedInstanceState);
037         mChannel=new ChannelDataHelper(this);
038         //取得数据库所有内容
039         channelList=mChannel.GetChannelList();
040         mAdapter=new ChannelAdapter(this);
041         //数据库无内容
042         if(mAdapter.getCount() == 0)
043             Toast.makeText(this, "尚未添加任何频道", Toast.LENGTH_SHORT)
            .show();
044         setTitle("频道选择");
045         //设置适配器
046         setListAdapter(mAdapter);
047
048     }
049     //ListView 是适配器
050     class ChannelAdapter extends BaseAdapter  {
051         private LayoutInflater mInflator;
052         //构造函数
053         public ChannelAdapter(Context context) {
054             this.mInflator = LayoutInflater.from(context);
055         }
056         //取得数组大小
057         public int getCount() {
058             return channelList.size();
059         }
```

```
060         //取得当前位置的元素
061         public String getItem(int position) {
062             return channelList.get(position);
063         }
064         //取得当前位置的 id
065         public long getItemId(int position) {
066             return 0;
067         }
068         //取得当前位置的视图
069         public View getView(int position, View convertView, ViewGroup
            parent) {
070             ViewHolder vRss = null;
071             //记录当前的位置
072             final int row = position;
073             //初始化界面元素类
074             vRss = new ViewHolder();
075             //膨胀出界面
076             convertView = mInflator.inflate(R.layout.item, null);
077             //取得界面元素
078             vRss.textLayout=(LinearLayout)convertView.findViewById
            (R.id.textLayout);
079             vRss.channel = (TextView)convertView.findView
            ById(R.id.title);
080             //设置标题字体大小
081             vRss.channel.setTextSize(25);
082             vRss.delBtn = (Button)convertView.findView
            ById(R.id.del_btn);
083             //取得标题的文本内容
084             String title = channelList.get(position);
085             vRss.channel.setText(title);
086             Log.e("guojs",channelList.get(row));
087             //设置按键监听器
088             vRss.textLayout.setOnClickListener(new OnClickListener(){
089                 @Override
090                 public void onClick(View v) {
091                     // TODO Auto-generated method stub
092                     String channelName=channelList.get(row);
093                     Intent it=new Intent();
094                     Log.e("channel-url",mChannel.getUrlByChannel(chan
                    nelName));
095                     //传递当前单击 RSS 源对应的 URL 地址
096                     it.putExtra("channel", mChannel.getUrlByChannel (cha
                    nnelName));
097                     it.setClass(SelectChannel.this, ActivityMain. class);
098                     //启动显示当前 RSS 源对应信息的界面
099                     startActivity(it);
100                 }
101             });
102             //删除当前 RSS 源
103             vRss.delBtn.setOnClickListener(new OnClickListener() {
104                 public void onClick(View v) {
105                     delRssInfo();
106                 }
107                 private void delRssInfo() {
108                     //删除成功
109                     if(-1 != mChannel.DelChannelInfo(channelList.
                    get(row)))
110                     {
111                         Toast.makeText(SelectChannel.this, "删除成功! ",
```

```
                            Toast.LENGTH SHORT).show();
112                         //移除数组元素
113                         channelList.remove(row);
114                         //通知改变界面
115                         mAdapter.notifyDataSetChanged();
116                         //删除失败
117                     }else{
118                         Toast.makeText(SelectChannel.this, "删除失败！",
                            Toast.LENGTH_SHORT).show();
119                     }
120
121                 }
122             });
123             return convertView;
124         }
125     }
126     //每一行元素的界面元素组成
127     final class ViewHolder {
128         public LinearLayout textLayout;
129         public TextView channel;
130         public Button delBtn;
131     }
132
133     //创建 menu 菜单
134     @Override
135     public boolean onCreateOptionsMenu(Menu menu) {
136         //添加 RSS 源
137         menu.add(0, MENU_ADD, 0, R.string.add_rss);
138         //退出程序
139         menu.add(0,MENU QUIT,1,R.string.rss quit);
140         return super.onCreateOptionsMenu(menu);
141     }
142     //为 menu 菜单按键绑定监听器
143     @Override
144     public boolean onOptionsItemSelected(MenuItem item) {
145         switch (item.getItemId()) {
146         //添加 RSS 源
147         case MENU ADD:
148             Intent intent = new Intent();
149             intent.setClass(SelectChannel.this, AddRss.class);
150             startActivity(intent);
151             return true;
152             //退出程序
153         case MENU_QUIT:
154             SelectChannel.this.finish();
155         default:
156             break;
157         }
158         return super.onOptionsItemSelected(item);
159     }
160 }
```

14.3.2　创建数据库

在包 com.rss.data 中新建 RSS 源信息数据库操作类 ChannelDataHelper，代码如下所示，数据库文件名称为 RssChannel.db，在构造函数中实例化数据库帮助类，并获得当前数据库。

数据库帮助类的实现在代码 81~111 行，第一次调用数据库帮助类的时候，将执行 onCreate()函数，新建表 channel，表中包含的字段有 id、name、url。

在该数据库操作类中需要实现的函数有 GetChannelList()、getUrlByChannel()、SaveChannelInfo()和 DelChannelInfo()，分别用于获取所有数据库信息、通过 RSS 源名称获得对应的 URL 地址、保存 RSS 源信息和删除 RSS 源信息。

```
001 package com.rss.data;                //声明包语句
//002~011 行为引入相关类，这里不再列举，请阅读光盘内容
……
012 //RSS 数据库操作类
013 public class ChannelDataHelper {
014         //数据库名称
015         private static String DB_NAME = "RssChannel.db";
016         //数据库版本
017         private static int DB VERSION = 1;
018         private SQLiteDatabase db;
019         private SqliteHelper dbHelper;
020         //构造函数
021         public ChannelDataHelper(Context context){
022             //实例化数据库帮助类
023           dbHelper=new SqliteHelper(context,DB_NAME, null, DB_VERSION);
024             //获得当前数据库
025             db= dbHelper.getWritableDatabase();
026         }
027         //关闭数据库
028         public void Close()
029         {
030             db.close();
031             dbHelper.close();
032         }
033         //获取所有的 RSS 源信息
034         public List<String> GetChannelList()
035         {
036             List<String> ChannelList = new ArrayList<String>();
037             //获得数据表中的所有数据
038             Cursor cursor=db.query(SqliteHelper.TB NAME, null, null,
                null, null, null, ID+" DESC");
039             //将游标移动到开始
040             cursor.moveToFirst();
041             //循环遍历整个数据库
042             while(!cursor.isAfterLast()&& (cursor.getString(1)!
                =null)){
043                 //取得源标题信息
044                 String channel=cursor.getString(1);
045                 //添加到数组
046                 ChannelList.add(channel);
047                 //移动游标到下一个元素
048                 cursor.moveToNext();
049             }
050             cursor.close();
051             return ChannelList;
052         }
053         //通过 url 名称获得 RSS 源 url 地址
054         public String getUrlByChannel(String name)
055         {
056             Cursor cursor=db.query(SqliteHelper.TB_NAME, null,
```

```
            NAME+"=?", new String[]{name}, null, null, null);
057             //将游标移动到开始
058             cursor.moveToFirst();
059             if(!cursor.isAfterLast()&& (cursor.getString(1)!=null))
060                 return cursor.getString(2);
061             return null;
062         }
063         //添加记录
064         public Long SaveChannelInfo(String name,String url)
065         {
066             ContentValues values = new ContentValues();
067             values.put(NAME, name);
068             values.put(URL, url);
069             //插入新纪录
070             Long id = db.insert(SqliteHelper.TB NAME, null, values);
071             Log.e("SaveChannelInfo",id+"");
072             return id;
073         }
074         //删除指定 channle 的记录
075         public int DelChannelInfo(String name){
076             int id= db.delete(SqliteHelper.TB_NAME, NAME +"='"+name+"'",
                null);
077             Log.e("DelChannelInfo",id+"");
078             return id;
079         }
080         //数据表的列信息
081     static String ID="id";
082     static String NAME="name";
083     static String URL="url";
084     //数据库帮助类
085     class SqliteHelper extends SQLiteOpenHelper{
086 
087         //用来保存 RSS 频道的表名
088         public static final String TB_NAME="channel";
089         public SqliteHelper(Context context, String name, CursorFactory
            factory, int version) {
090         super(context, name, factory, version);
091         }
092         //创建表
093         @Override
094         public void onCreate(SQLiteDatabase db) {
095             db.execSQL("CREATE TABLE IF NOT EXISTS "+
096             TB NAME+"("+
097             ID+" integer primary key,"+
098             NAME+" varchar,"+
099             URL+" varchar"+
100             ")"
101             );
102             Log.e("Database","onCreate");
103         }
104         //更新表
105         @Override
106         public void onUpgrade(SQLiteDatabase db, int oldVersion, int
            newVersion) {
107             db.execSQL("DROP TABLE IF EXISTS " + TB NAME);
108             onCreate(db);
109             Log.e("Database","onUpgrade");
```

```
110         }
111     }
```

14.3.3　显示每行元素的界面

用于显示 RSS 每行元素的界面如下所示，包括文本区域和按键区域，文本区域又包括标题和日期，按键区域为一个“删除”按钮。这个界面是与另一个界面 RSS 源阅读界面共用的，因此有一个日期的文本区域，我们不用去赋值即可。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
03     android:orientation="horizontal"
04     android:layout_width="fill_parent"
05     android:layout_height="fill_parent"
06     >
07     <!-- 文本区域 -->
08     <LinearLayout
09           android:id="@+id/textLayout"
10           android:orientation="vertical"
11           android:layout width="270dp"
12           android:layout height="fill parent">
13           <!-- 标题  -->
14           <TextView android:id="@+id/title"
15              android:layout width="wrap content"
16              android:layout height="wrap content"
17              android:textColor="#FFFFFFFF"
18              android:textSize="15px" />
19           <!-- 日期 -->
20           <TextView android:id="@+id/pubdate"
21              android:layout width="wrap content"
22              android:layout height="wrap content"
23              android:textColor="#FFFFFFFF"
24              android:textSize="10px" />
25        </LinearLayout>
26        <!-- 删除按钮 -->
27        <Button android:id="@+id/del_btn"
28           android:layout_width="50dp"
29           android:layout_height="wrap_content"
30           android:text="删除"
31           android:layout gravity="right" />
32
33  </LinearLayout>
```

14.3.4　添加 RSS 源界面

如图 14.4 所示，为添加 RSS 源的界面。界面实现代码如下所示，最上面一个 TextView 用于显示标题，中间一个 EditText 用于给用户输入 RSS 源的网址。最下面为 3 个按钮，分别用于检查网址是否为有效的 RSS 网址、添加 RSS 源到数据库、退出添加界面。



图 14.4　RSS 源添加界面

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com
    /apk/res/android"
03     android:orientation="vertical"
04     android:layout width="fill parent"
05     android:layout height="fill parent"
06     >
07     <!-- 标题 -->
08     <TextView
09        android:id="@+id/show_title"
10        android:layout width="fill parent"
11        android:layout_height="wrap_content"
12        android:text="@string/title show"
13        />
14     <!-- RSS 源地址 -->
15     <EditText
16        android:layout marginTop="3dp"
17        android:id="@+id/rss_add"
18        android:layout_width="fill_parent"
19        android:layout_height="wrap_content"
20        android:text=""/>
21     <!-- 按键布局 -->
22     <LinearLayout
23        android:layout marginTop="30dp"
24        android:orientation="horizontal"
25        android:layout width="fill parent"
26        android:layout height="wrap content">
27        <!--"验证"按钮 -->
28        <Button
29           android:id="@+id/verify_rss"
30           android:layout_width="80dp"
31           android:layout_height="wrap_content"
32           android:layout_marginLeft="5dp"
33           android:text="@string/rss verify"
34           />
35        <!--"添加"按钮 -->
36        <Button
37           android:id="@+id/add rss"
38           android:layout_width="80dp"
```

```
39              android:layout height="wrap content"
40              android:layout marginLeft="5dp"
41              android:text="@string/rss add"
42              />
43          <!-- 退出程序 -->
44          <Button
45              android:id="@+id/quit_rss"
46              android:layout_width="80dp"
47              android:layout_height="wrap_content"
48              android:layout_marginLeft="5dp"
49              android:text="@string/rss_quit"
50              />
51      </LinearLayout>
52  </LinearLayout>
```

14.3.5　实现添加 RSS 源界面的功能

在包 com.rss.activity 中新建 AddRss.java，用于实现添加 RSS 源界面的功能。这个界面主要实现 3 个功能，分别为检查 RSS 源的有效性、添加 RSS 源信息到数据库和退出当前界面。在 onCreate()函数中实例化界面的元素，为按键绑定监听器，同时实例化 RSS 源数据库操作类。

当用户单击“检验”按钮时将调用 verifyRss()函数用于检验当前输入的 url 地址是否为有效的 RSS 网址。verifyRss()函数的实现如以下代码 098~119 行所示，采用 SAX 方式解析网址内容，新建 SAX 处理类 getRssChannel 继承于 DefaultHandler。因为我们只要知道当前的源的名称，根据前面我们分析的 RSS 2.0 的 xml 结构，这个源的名称应该在 channel 标签里面的 title 标签，并且这个 title 标签不在 item 中。基于这个分析我们设置标志变量 flag，当遇到 channel 标签则将标志位置 1，在没有遇到 item 标签之前遇到 title 标签则将标志位置为 2。这里的 title 标签就是我们需要解析的标签，将此标签的内容保存到 channel 中。同时一旦遇到 item 标签，则将标志位置为 0。

通过验证 verifyRss()函数返回的值就可以判断当前 RSS 是否有效，当然，还不是那么严谨，读者可以想一下为什么不够严谨，如何更严谨地验证，这里作为演示不再赘述。验证结果将会通过 Toast 的方式显示出来。

当用户单击“保存”按钮，首先验证 RSS 源的网址是否有效，并获得当前 RSS 源的标题。接着通过 RSS 源数据库操作类调用函数 SaveChannelInfo 保存数据到数据库中。

```
001 package com.rss.activity;                //声明包语句
002~021 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
022      //RSS 源添加界面
023 public class AddRss extends Activity {
024     //"添加"按钮
025     private Button addRss;
026     //"检验"按钮
027     private Button verifyRss;
028     //"退出"按钮
029     private Button quit;
030     //RSS 源地址
031     private EditText rssText;
```

```
032     //RSS 源数据库操作类
033     ChannelDataHelper mChannelData;
034     Context mContext;
035     @Override
036     protected void onCreate(Bundle savedInstanceState) {
037         // TODO Auto-generated method stub
038         super.onCreate(savedInstanceState);
039         setContentView(R.layout.add_rss_dialog);
040         mContext=this;
041         //实例化 RSS 源数据库操作类
042         mChannelData=new ChannelDataHelper(this);
043         //初始化界面元素
044         addRss = (Button)findViewById(R.id.add_rss);
045         verifyRss = (Button)findViewById(R.id.verify_rss);
046         quit = (Button)findViewById(R.id.quit_rss);
047         rssText = (EditText)findViewById(R.id.rss_add);
048         //设置按键监听器
049         addRss.setOnClickListener(clickListener);
050         verifyRss.setOnClickListener(clickListener);
051         quit.setOnClickListener(clickListener);
052     }
053     //按键监听器
054     OnClickListener clickListener = new OnClickListener() {
055         @Override
056         public void onClick(View v) {
057             switch (v.getId()) {
058             //"添加"按钮
059             case R.id.add rss:
060                 saveRss();
061                 break;
062             //检验地址
063             case R.id.verify_rss:
064                 if(verifyRss(rssText.getText().toString()) != null
065                     && verifyRss(rssText.getText().toString()) != "默认")
066                     Toast.makeText(mContext, "验证通过", Toast.LENGTH_
                         SHORT).show();
067                 else
068                     Toast.makeText(mContext, "验证失败", Toast.LENGTH_
                        SHORT).show();
069                 break;
070             //退出添加界面
071             case R.id.quit_rss:
072                 Intent it=new Intent();
073                 it.setClass(AddRss.this, SelectChannel.class);
074                 //进入 RSS 源选择界面
075                 startActivity(it);
076                 AddRss.this.finish();
077                 break;
078             default:
079                 break;
080             }
081
082         }
083     };
084     //保存 RSS 源
085     protected void saveRss() {
086         String rssAddress = rssText.getText().toString().trim();
087         String rssName= verifyRss(rssAddress);
088         //RSS 源地址正确并正确保存
```

```
089         if(rssName != null && rssName != "默认" && (-1 != mChanne
            lData.SaveChannelInfo(rssName, rssAddress)))
090         {
091             Toast.makeText(mContext, "保存成功!", Toast.LENGTH_
                SHORT).show();
092             rssText.setText("");
093         }else{
094             Toast.makeText(mContext, "保存失败! ", Toast.LENGTH SHORT)
            .show();
095         }
096     }
097     //验证 RSS 源地址的正确性，返回 RSS 源的标题
098     private String verifyRss(String urlString) {
099         try {
100                URL url = new URL(urlString);
101                //实例化 SAX 解析工厂类
102                SAXParserFactory factory = SAXParserFactory.newInstance();
103                //SAX 解析器
104                SAXParser parser = factory.newSAXParser();
105                XMLReader xmlReader = parser.getXMLReader();
106                //SAX 处理类
107                getRssChannel rssHandler = new getRssChannel();
108                xmlReader.setContentHandler(rssHandler);
109                //取得 url 网址内容
110                InputSource is = new InputSource(url.openStream());
111                //解析 url 网址内容
112                xmlReader.parse(is);
113                Log.e("channelName",rssHandler.getChannel());
114                return rssHandler.getChannel();
115         }catch (Exception e) {
116             Log.e("channnelName",e.toString());
117             return null;
118         }
119     }
120     class getRssChannel extends DefaultHandler {
121         //标志位，用于表示 RSS 源的位置
122         int flag=1;
123         String channel="默认";
124         public getRssChannel() {
125         }
126         //返回 channel 信息
127         public String getChannel()
128         {
129             return channel;
130         }
131         //文档开始
132         @Override
133         public void startDocument()throws SAXException {
134         }
135         //文档结尾
136         @Override
137         public void endDocument() throws SAXException {
138             super.endDocument();
139         }
140         //开始解析标签
141         @Override
142         public void startElement(String uri, String localName, String
             qName,
143                 Attributes attributes) throws SAXException {
```

```
144             super.startElement(uri, localName, qName, attributes);
145             if(localName.equals("channel")) {
146                 flag = 1;
147                 return;
148             }
149             //一旦进入了 item，则将标志为设置为 0
150             if(localName.equals("item")) {
151                 flag=0;
152                 return;
153             }
154             //在 item 外面遇到 title 标签，将标志位设置为 2
155             if(flag == 1)
156             {
157                 if(localName.equals("title"))
158                 {
159                     flag = 2;
160                 }
161             }
162         }
163         //标签结束
164         @Override
165         public void endElement(String uri, String localName, String q
        Name)
166                 throws SAXException {
167         }
168         //解析标签内容
169         @Override
170         public void characters(char[] ch, int start, int length)
171                 throws SAXException {
172             if(length <= 5) return ;
173             String theString = new String(ch, start, length);
174             //当标志位为 2 表明当前正在解析 RSS 源标题的标签
175             if(flag == 2)
176             {
177                 //保存 RSS 源标题
178                 Log.e("channel",theString);
179                 channel=theString;
180                 flag =0;
181             }
182         }
183     }
184 }
```

14.4　读取 RSS 源

14.4.1　存放 RSS 信息

为了保存解析到的 RSS 信息，首先需要新建一个类 RssItem 用于存放 RSS 信息，代码如下所示，首先需要新建一个类 RssItem 用于存放 RSS 信息，在包 com.rss.data 中新建 RssItem.java。代码如下所示，设置 RSS 信息对应的几个成员变量，如标题、详细信息、链接、日期等。

```
01  package com.rss.data;
02  //Rssitem 信息，用于存放 RSS 的每一个 item
```

```
03  public class RssItem {
04      public static final String TITLE = "title";
05      public static final String PUBDATE = "pubDate";
06      //标题
07      private String title;
08      //详细内容
09      private String description;
10      //链接
11      private String link;
12      //来源
13      private String source;
14      //日期
15      private String pubDate;
16      //获得标题
17      public String getTitle() {
18          return title;
19      }
20      //设置标题
21      public void setTitle(String title) {
22          this.title = title;
23      }
24      //取得详细信息
25      public String getDescription() {
26          return description;
27      }
28      //设置详细信息
29      public void setDescription(String description) {
30          this.description = description;
31      }
32      //取得链接
33      public String getLink() {
34          return link;
35      }
36      //设置链接
37      public void setLink(String link) {
38          this.link = link;
39      }
40      //取得来源
41      public String getSource() {
42          return source;
43      }
44      //设置来源
45      public void setSource(String source) {
46          this.source = source;
47      }
48      //取得日期
49      public String getPubDate() {
50          return pubDate;
51      }
52      //设置日期
53      public void setPubDate(String pubDate) {
54          this.pubDate = pubDate;
55      }
56      //转化成 string
57      public String toString() {
58          if(title.length() > 20) {
59              return title.substring(0, 42) + "...";
60          }else {
61              return title;
```

```
62          }
63      }
64  }
```

14.4.2　取出需要的信息

因为主页面中只需要显示 RSS 的标题和日期，因此我们需要对所获得的 RSS 信息再进行加工，提取出需要的信息。如下所示，通过传入一个由 RssItem 组成的数组，可以转化为元素为 HashMap<String,Object>的 List。

```
01  package com.rss.data;            //声明包语句
02~08 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
09  public class RssFeed {
10      //标题
11      private String title;
12      //日期
13      private String pubdate;
14      //数量
15      private int itemcount;
16      //用于存放 RSS 信息的列表
17      private List<RssItem> itemlist;
18      public RssFeed() {
19          itemlist = new Vector<RssItem>(0);
20      }
21      //添加元素
22      public int addItem(RssItem item) {
23          itemlist.add(item);
24          itemcount++;
25          return itemcount;
26      }
27      //取得元素
28      public RssItem getItem(int location) {
29          return itemlist.get(location);
30      }
31      //取得所有列表需要的信息，存储到 List
32      public List getAllItemsForListView() {
33          List<Map<String, Object>> data = new ArrayList<Map<String,
            Object>>();
34          int size = itemlist.size();
35          for(int i=0; i<size; i++) {
36              HashMap<String, Object> item = new HashMap<String, Object>();
37              item.put(RssItem.TITLE, itemlist.get(i).getTitle());
38              item.put(RssItem.PUBDATE, itemlist.get(i).getPubDate());
39              data.add(item);
40          }
41
42          return data;
43      }
44      //取得标题
45      public String getTitle() {
46          return title;
47      }
48      //设置标题
49      public void setTitle(String title) {
```

```
50          this.title = title;
51      }
52      //取得日期
53      public String getPubdate() {
54          return pubdate;
55      }
56      //设置日期
57      public void setPubdate(String pubdate) {
58          this.pubdate = pubdate;
59      }
60      //取得数量
61      public int getItemcount() {
62          return itemcount;
63      }
64      //设置数量
65      public void setItemcount(int itemcount) {
66          this.itemcount = itemcount;
67      }
68      //获得信息数组
69      public List<RssItem> getItemlist() {
70          return itemlist;
71      }
72      //设置信息数组
73      public void setItemlist(List<RssItem> itemlist) {
74          this.itemlist = itemlist;
75      }
76  }
```

14.4.3　对 XML 文件进行解析

最关键的部分就是对 XML 文件进行解析，在包 com.rss.sax 下新建 RssHandler.java，代码如下所示，新建类 RssHandler 继承于 DefaultHandler。根据一般的 SAX 解析流程，依次解析文档头、标签头、文本、标签尾、文档尾，我们主要解析的是数据中每个<item>标签内的内容，将<item>标签内的内容逐个提取出来存储到 RssItem 类中，并传入类 RssFeed 中进行加工，最终得到我们需要的数据形式。

```
001 package com.rss.sax;                //声明包语句
002~011 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
012 //RSS 数据 SAX 解析
013 public class RssHandler extends DefaultHandler {
014     RssFeed rssFeed;
015     RssItem rssItem;
016     private static boolean a = false;
017     String lastElementName = "";
018     final int RSS_TITLE = 1;
019     final int RSS_LINK = 2;
020     final int RSS_DESCRIPTION = 3;
021     final int RSS_PUBDATE = 5;
022     int currentstate = 0;
023
024     public RssHandler() {
025
026     }
```

```
027     //返回 rssFeed 形式的信息
028     public RssFeed getFeed() {
029         return rssFeed;
030     }
031     //开始解析文档
032     public void startDocument()throws SAXException {
033         System.out.println("startDocument");
034         //实例化 RssFeed 类和 RssItem 类
035         rssFeed = new RssFeed();
036         rssItem = new RssItem();
037     }
038     //文档结束
039     @Override
040     public void endDocument() throws SAXException {
041         super.endDocument();
042     }
043     //开始解析标签
044     @Override
045     public void startElement(String uri, String localName, String qName,
046             Attributes attributes) throws SAXException {
047         super.startElement(uri, localName, qName, attributes);
048         //channel 标签
049         if(localName.equals("channel")) {
050                 currentstate = 0;
051                 return;
052         }
053         //item 标签
054         if(localName.equals("item")) {
055             rssItem = new RssItem();
056             return;
057         }
058         //title 标签
059         if(localName.equals("title")) {
060             currentstate = RSS TITLE;
061             return;
062         }
063         //详细信息标签
064         if(localName.equals("description")) {
065                 //跳过第一次遇到的 description 标签
066             if(a == true) {
067                 currentstate = RSS DESCRIPTION;
068                 return;
069             } else {
070                 a = true;
071                 return;
072             }
073         }
074         //链接
075         if(localName.equals("link")) {
076             currentstate = RSS LINK;
077             return;
078         }
079         //日期
080         if(localName.equals("pubDate")) {
081             currentstate = RSS PUBDATE;
082             return;
083         }
084         currentstate = 0;
085     }
```

```
086     //标签结束
087     @Override
088     public void endElement(String uri, String localName, String qName)
089             throws SAXException {
090         //将信息添加到 rssFeed 中
091         if(localName.equals("item")) {
092             rssFeed.addItem(rssItem);
093             return;
094         }
095     }
096     //解析标签间的文本
097     @Override
098     public void characters(char[] ch, int start, int length)
099             throws SAXException {
100         if(length <= 5) return ;
101         String theString = new String(ch, start, length);
102         switch (currentstate) {
103         //标题
104         case RSS_TITLE:
105             rssItem.setTitle(theString);
106             Log.e("title",theString);
107             currentstate = 0;
108             break;
109             //链接
110         case RSS_LINK:
111             rssItem.setLink(theString);
112             currentstate = 0;
113             break;
114             //详细信息
115         case RSS_DESCRIPTION:
116             System.out.println("RSS_DESCRIPTION=" + theString);
117             if(a == true) {
118                 rssItem.setDescription(theString);
119                 currentstate = 0;
120             }else {
121                 a = true;
122             }
123             break;
124             //日期
125         case RSS_PUBDATE:
126             rssItem.setPubDate(theString);
127             currentstate = 0;
128             break;
129         default:
130             return;
131         }
132     }
133 }
```

14.4.4　调用、共享 RSS 信息

为了给其他程序调用、共享 RSS 的信息，我们需要新建一个 ContentProvider，在包 com.rss.db 中新建 RssProvider.java。代码如下所示，新建类 RssProvider 继承于 ContentProvider，在代码 60～79 行新建一个内部类 RssDatabaseHelper 用于创建和升级数据库。在初始化函数中实例化该数据库操作类，并获得对应的数据库。

代码 049～058 行实现了一个 uri 匹配器，用于匹配 uri 地址是针对单个 RSS 信息进行操作还是对所有的 RSS 信息进行操作。接着在函数中依次实现了插入、查询、更新等操作，方便本程序和其他程序编辑数据库内容。

```
001 package com.rss.db;                    //声明包语句
002~014 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
015 //RSS 数据内容管理
016 public class RssProvider extends ContentProvider {
017     //RSS 内容数据库文件名
018     private static final String RSS_DATABASE = "rss.db";
019     //数据库表名
020     private static final String RSS_TABLE = "rss_item";
021     private static final int RSS_DATABASE_VERSION = 1;
022     private static final String TAG = "RssProvider";
023     //URI
024     public static final Uri RSS_URI = Uri.parse("content:
        //com.rss.activity/rss");
025
026     //列名
027     public static final String RSS_ID = " id";
028     public static final String RSS_TITLE = "title";
029     public static final String RSS_DESCRIPTION = "description";
030     public static final String RSS_LINK = "link";
031     public static final String RSS_PUBDATE = "pubDate";
032
033     //列的索引
034     public static final int TITLE_INDEX = 1;
035     public static final int DESCRIPTION_INDEX = 2;
036     public static final int LINK_INDEX = 3;
037     public static final int PUBDATE_INDEX = 4;
038
039     //创建表的 sql 语句
040     private static final String RSS_SQL = "CREATE TABLE " + RSS_TABLE
+ "("
041                 + RSS_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
042                 + RSS_TITLE + " TEXT, "
043                 + RSS_DESCRIPTION + " TEXT, "
044                 + RSS_LINK + " TEXT, "
045                 + RSS_PUBDATE + " DATE);";
046
047     SQLiteDatabase rssDb = null;
048
049     //创建用来区分不同 URI 的常量
050     private static final int RSS = 1;
051     private static final int RSSID = 2;
052     //uri 地址适配器
053     private static UriMatcher uriMatcher;
054     static {
055         uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
056         uriMatcher.addURI("com.rss.activity", "rss", RSS);
057         uriMatcher.addURI("com.rss.activity", "rss/#", RSSID);
058     }
059     //创建管理数据的 helper 类
060     private static class RssDatabaseHelper extends SQLiteOpenHelper {
061         //构造函数
062         public RssDatabaseHelper(Context context) {
```

```
063             super(context, RSS DATABASE, null, RSS DATABASE VERSION);
064             Log.v(TAG, "Rss database create successfully!");
065         }
066         //第一次创建数据库时调用
067         @Override
068         public void onCreate(SQLiteDatabase db) {
069             db.execSQL(RSS_SQL);
070             Log.v(TAG, "Rss table create successfully!");
071         }
072         //升级时调用
073         @Override
074         public void onUpgrade(SQLiteDatabase db, int oldVersion, int
            newVersion) {
075             db.execSQL("DROP TABLE IF EXISTS " + RSS TABLE);
076             onCreate(db);//重新创建表
077         }
078 
079     }
080     @Override
081     public int delete(Uri uri, String where, String[] whereArgs) {
082         return 0;
083     }
084     //取得 uri 地址对应的操作
085     @Override
086     public String getType(Uri uri) {
087         switch (uriMatcher.match(uri)) {
088         case RSS:
089             return "vnd.android.cursor.dir/com.rss.activity";
090         case RSSID:
091             return "vnd.android.cursor.item/com.rss.activity";
092         default:
093             throw new IllegalArgumentException("Unsupport URI:" + uri);
094         }
095     }
096     //存储数据
097     @Override
098     public Uri insert(Uri  uri, ContentValues values) {
099         Uri uri = null;
100         long rowId = rssDb.insert(RSS TABLE, null, values);
101         //返回的 rowId > 0
102         if(rowId > 0) {
103             uri = ContentUris.withAppendedId(RSS_URI, rowId);
104             getContext().getContentResolver().notifyChange(uri, null);
105         }
106         return uri;
107     }
108     //初始化函数
109     @Override
110     public boolean onCreate() {
111         Context context = getContext();
112         //实例化数据库帮助类
113         RssDatabaseHelper rssDbHelper = new RssDatabaseHelper(context);
114         //获得对应的是数据库
115         rssDb = rssDbHelper.getWritableDatabase();
116         return (rssDb == null) ? false : true;
117     }
118     //查询
119     @Override
120     public Cursor query(Uri uri, String[] projection, String selection,
121             String[] selectionArgs, String sortOrder) {
```

```
122         SQLiteQueryBuilder sqb = new SQLiteQueryBuilder();
123         sqb.setTables(RSS_TABLE);//设置查询的表
124         //筛选 uri 地址
125         switch (uriMatcher.match(uri)) {
126         //查询单个 RSS 信息
127         case RSSID:
128             sqb.appendWhere(RSS_ID + "=" + uri.getPathSegments().
                get(1));
129             break;
130         default:
131             break;
132         }
133         String orderBy;
134         if(TextUtils.isEmpty(sortOrder)) {
135             orderBy = RSS_PUBDATE;
136         }else {
137             orderBy = sortOrder;
138         }
139         //对底层数据库的应用查询
140         Cursor c = sqb.query(rssDb, projection, selection, selectionArgs,
            null, null, orderBy);
141         c.setNotificationUri(getContext().getContentResolver(), uri);
142         return c;
143     }
144     //更新数据
145     @Override
146     public int update(Uri uri, ContentValues values, String selection,
147             String[] selectionArgs) {
148         int count;
149         //匹配 uri 地址
150         switch (uriMatcher.match(uri)) {
151         case RSS:
152         count = rssDb.update(RSS_TABLE, values, selection, selection
            Args);
153             break;
154         case RSSID:
155             String segment = uri.getPathSegments().get(1);
156             count = rssDb.update(RSS_TABLE, values, RSS_ID + "=" + segment
157                     + (!TextUtils.isEmpty(selection) ? " AND ("
158                     + selection +')' : ""), selectionArgs);
159             break;
160         default:
161             throw new IllegalArgumentException("Unkown URI " + uri);
162         }
163         //通知更改
164         getContext().getContentResolver().notifyChange(uri, null);
165         return count;
166     }
167 }
```

14.4.5　查看界面

如图 14.5 所示，为 RSS 信息查看界面。当用户单击任何一个 RSS 信息源时，程序将连接对应的网址，获取网页的数据，解析网页的内容。代码如下所示，通过 getExtras()获得前一个页面传递过来的数据，通过 getString(“channel”)获得 RSS 的 url 地址，接着调用函数 getFeed 获得解析 RSS 信息类，最后调用函数 showListView()将 RSS 信息显示到列表中。

getFeed()函数采用 SAX 解析方式解析 XML 内容，通过实例化 RssHandler 类来处理 XML 内容，并将解析的结果存储到 RssFeed 类中返回。showListView()函数通过 RSS 解析得到的 feed 变量获得信息的列表，接着将数据适配到列表中，为列表设置适配器。类 MyAdapter 用于为当前界面适配数据，在 getView()函数中为每一行对应的位置适配数据，并设置按键监听器。当单击任何一行时，将进入界面 ActivityShowDescription，并传递指定行的 RSS 信息。



图 14.5　RSS 信息

```
001 package com.rss.activity;                    //声明包语句
002~024 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
025 //RSS 文本解析
026 public class ActivityMain extends ListActivity {
027     //RSS 源地址
028     public String RSS_URL = "";
029     public final String TAG = "RssReader";
030     //RSS 信息类
031     private RssFeed feed;
032     //用于存放 RSS 信息的列表
033     private List<Map<String, Object>> mList;
034     //初始化函数
035     @Override
036     public void onCreate(Bundle savedInstanceState) {
037         super.onCreate(savedInstanceState);
038         Bundle b=getIntent().getExtras();
039         //取得 RSS 地址
040         RSS_URL=b.getString("channel");
041         Log.e("guojs-->get url",RSS_URL);
042         //取得对应 RSS 源地址的信息
043         feed = getFeed(RSS_URL);
044         //将 RSS 信息显示到列表中
045         showListView();
046     }
047     //取得对应 RSS 源地址的信息
```

```
048      private RssFeed getFeed(String urlString) {
049          try {
050              URL url = new URL(urlString);
051              //采用 SAX 解析
052              SAXParserFactory factory = SAXParserFactory.newInstance();
053              SAXParser parser = factory.newSAXParser();
054              XMLReader xmlReader = parser.getXMLReader();
055              //SAX 解析类
056              RssHandler rssHandler = new RssHandler();
057              xmlReader.setContentHandler(rssHandler);
058              //取得网页信息
059              InputSource is = new InputSource(url.openStream());
060              //解析网页内容
061              xmlReader.parse(is);
062              //返回 RSS 信息
063              return rssHandler.getFeed();
064          }catch (Exception e) {
065              return null;
066          }
067      }
068      //显示 RSS 信息到列表中
069      @SuppressWarnings("unchecked")
070      private void showListView() {
071          if(feed == null) {
072              setTitle("RSS 阅读器");
073              return;
074          }
075          //取得 RSS 信息数组
076          mList = feed.getAllItemsForListView();
077          MyAdapter adapter = new MyAdapter(this);
078          //设置适配器
079          setListAdapter(adapter);
080          setSelection(0);
081      }
082      //新建数据适配器类
083      class MyAdapter extends BaseAdapter  {
084          private LayoutInflater mInflator;
085          //构造函数
086          public MyAdapter(Context context) {
087              this.mInflator = LayoutInflater.from(context);
088          }
089          //取得列表数量
090          public int getCount() {
091              System.out.println("mList.size()=" + mList.size());
092              return mList.size();
093          }
094          public Object getItem(int position) {
095              return null;
096          }
097          //取得 item 的 id
098          public long getItemId(int position) {
099              return 0;
100          }
101          //取得每一行元素的视图
102          public View getView(int position, View convertView, ViewGroup
             parent) {
103              ViewRss vRss = null;
104              final int row = position;
```

```
105            //实例化界面元素类
106            vRss = new ViewRss();
107            //膨胀出界面元素的视图
108            convertView = mInflator.inflate(R.layout.item, null);
109            //初始化界面元素
110            vRss.mLayout=(LinearLayout)convertView.findViewById(R.id.
               textLayout);
111            vRss.title = (TextView)convertView.findViewById(R.
               id.title);
112            vRss.pubdate = (TextView)convertView.findViewById(R.
               id.pubdate);
113            vRss.delBtn = (Button)convertView.findViewById(R.
               id.del btn);
114            //设置"删除"按钮不可见
115            vRss.delBtn.setVisibility(View.INVISIBLE);
116            convertView.setTag(vRss);
117            //取得日期
118            String pubDate =  (String) mList.get(position).
               get("pubDate");
119            //取得标题
120            String title = (String)mList.get(position).get("title");
121            //设置标题
122            vRss.title.setText(title);
123            //设置日期
124            vRss.pubdate.setText(pubDate);
125            //为 textLayout 设置按键监听器
126            vRss.mLayout.setOnClickListener(new OnClickListener() {
127                public void onClick(View v) {
128                    Intent intent = new Intent(ActivityMain.this, Activity
                      ShowDescription.class);
129                    Bundle b = new Bundle();
130                    //绑定数据
131                    b.putString("title", feed.getItem(row).getTitle());
132                    b.putString("description", feed.getItem(row).
                      getDescription());
133                    b.putString("link", feed.getItem(row).getLink());
134                    b.putString("pubdate", feed.getItem(row).
                      getPubDate());
135                    intent.putExtra("com.rss.data.RssFeed", b);
136                    //启动查看 RSS 详细信息界面
137                    startActivity(intent);
138                }
139            });
140            return convertView;
141        }
142    }
143    //RSS 界面元素类，用于存放 RSS 界面元素
144    public final class ViewRss {
145        public LinearLayout mLayout;
146        //标题
147        public TextView title;
148        //日期
149        public TextView pubdate;
```

```
150         //"删除"按钮
151         public Button delBtn;
152     }
```

14.4.6　详细查看 RSS 信息

当用户单击任何一行 RSS 信息时，将进入 RSS 详细信息查看页面，如图 14.5 所示。在包 com.rss.activity 中新建 ActivityShowDescription.java 用于实现该界面功能，如下代码所示，在 onCreate()函数中获得上一个页面通过 intent 传递过来的信息，包括详细信息、链接、日期等，最后以网页形式显示到页面中，如代码 50 行所示。此外，在最下面设置两个按钮，一个为“收藏”，一个为“返回”，当单击“收藏”按钮时将调用函数 storeDataRss()存储当前 RSS 的数据到数据库中；当单击“返回”按钮时，将关闭当前页面，显示上一个页面。



图 14.6　RSS 详细信息查看页面

```
01  package com.rss.activity;                  //声明包语句
02~14 行为引入相关类，这里不再列举，请阅读光盘内容
//
……
15  //RSS 详细信息显示
16  public class ActivityShowDescription extends Activity {
17      //标题
18      private String title = null;
19      //日期
20      private String pubdate = null;
21      //详细信息
22      private String description = null;
23      //链接
24      private String link = null;
25      //初始化函数
26      @Override
27      protected void onCreate(Bundle savedInstanceState) {
28          super.onCreate(savedInstanceState);
29          setContentView(R.layout.showdescription);
30          String content = null;
31          Intent intent = getIntent();
32          //取得 intent 携带的信息
33          if(intent != null) {
```

```
34          Bundle bundle = intent.getBundleExtra
            ("com.rss.data.RssFeed");
35          title = bundle.getString("title");
36          pubdate = bundle.getString("pubdate");
37          description = bundle.getString("description");
38          link = bundle.getString("link");
39          content ="内容: <br />" + description.replace('\n', ' ')
40          + "<br /><br />链接:\n<a href=\"" + link +"\">"+link+"</a>"
41          + "<br /><br />日期: " + pubdate;
42      }else {
43          content = "error!";
44      }
45      //设置当前界面的标题
46      ActivityShowDescription.this.setTitle(title);
47      //初始化页面元素
48      WebView textView = (WebView)findViewById(R.id.content);
49      //将 content 显示到网页中
50      textView.loadDataWithBaseURL("",content,"text/html","UTF-8
        ","");
51      Button backButton = (Button)findViewById(R.id.back);
52      Button storeButton = (Button)findViewById(R.id.store);
53      //为按键绑定监听器
54      storeButton.setOnClickListener(new OnClickListener() {
55          public void onClick(View v) {
56              //保存 RSS 数据到数据库中
57              storeDataRss();
58                          Toast.makeText(ActivityShowDescription.
                            this,"收藏成功! ",Toast.LENGTH_LONG).show();
59          }
60      });
61
62      backButton.setOnClickListener(new OnClickListener() {
63          //关闭当前页面
64          public void onClick(View v) {
65              ActivityShowDescription.this.finish();
66          }
67      });
68  }
69  //保存 RSS 数据到数据库中
70  protected void storeDataRss() {
71      ContentResolver cr = getContentResolver();
72      ContentValues values = new ContentValues();
73      values.put(RssProvider.RSS_TITLE, title);
74      values.put(RssProvider.RSS_DESCRIPTION, description);
75      values.put(RssProvider.RSS_PUBDATE, pubdate);
76      values.put(RssProvider.RSS_LINK, link);
77      cr.insert(RssProvider.RSS_URI, values);
78  }
```

14.5　知 识 拓 展

我们在分析 RSS 网页内容时经常可以看到以下代码 03 行所示的格式，为什么要在字符外面加上<![[CDATA]]>标签呢？

术语 CDATA 指的是不应由 XML 解析器进行解析的文本数据（Unparsed Character

Data），CDATA 部分中的所有内容都会被解析器忽略，CDATA 部分由 “<![CDATA[” 开始，由 “]]>” 结束。在 XML 元素中，“<” 和 “&” 是非法的。“<” 会产生错误，因为解析器会把该字符解释为新元素的开始。"&" 也会产生错误，因为解析器会把该字符解释为字符实体的开始。所以，为了防止解析器错误地解析 XML 中的内容，一般在内容区域加入 CDATA 标签。

```
01  <item>
02          <title>
03              <![CDATA[视频：非农造就空方强势]]>
04          </title>
05  <link>http://go.rss.sina.com.cn/redirect.php?url=http:
    //video.sina.com.cn/p/finance/stock/jsy/20121105/173961905847.html</
    link>
06          <author>WWW.SINA.COM.CN</author>
07  <guid>http://go.rss.sina.com.cn/redirect.php?url=http:
    //video.sina.com.cn/p/finance/stock/jsy/20121105/173961905847.html</
    guid>
08          <category>
09              <![CDATA[财经频道-股市及时雨]]>
10          </category>
```

14.6　本章小结

本章介绍了 RSS 新闻阅读器的开发。在这个信息膨胀的年代，如何让用户最快地获得有价值的信息是我们需要思考的方向。本程序主要采用了 SAX 方式解析 RSS 数据，并将数据以数据库的方式进行存储。读者需要注意的是，在解析 XML 文件时，需要先对整个 XML 文件结构了解清楚，才能准确解析到想要的数据。

第 15 章　Android 地图应用

在 Android 开发中地图是很多软件不可或缺的内容，本章将以百度地图为例，为大家讲解百度地图 API 的使用。

15.1　开发前准备

要使用百度地图 API，首先要到其网站上申请 Key，申请地址为 http://dev.baidu.com/wiki/static/imap/key/，然后进入如图 15.1 所示的界面，简单输入相应信息后单击“生成 API 密钥”即可生成 Key。



图 15.1　百度地图 Key 申请界面

单击“我的 Key”按钮，可以查看当前 Key 的列表，如图 15.2 所示。申请完毕之后还需要下载百度地图 API 的 jar 包，下载完之后打开下载包可以发现里面有一个 jar 包和一个.so 文件，需要把这两个文件复制到项目根目录下的 libs 文件夹下，并且还要新建一个 armeabi 文件夹，因为.so 文件需要放到这个文件夹下。

我的key列表

序号	key	应用名称	应用描述
1	5288097969F6ABF9F9C2FAAE9ED0C8F438BC4AFD	baidu地图测试	用于baidu地图api测试

图 15.2　地图 Key 列表

最后需要在 AndroidManifest.xml 文件中添加权限如下所示：

```
01  <uses-permission android:name="android.permission.
    ACCESS_NETWORK_STATE" />
02  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
03  <uses-permission android:name="android.permission.INTERNET" />
04  <uses-permission android:name="android.permission.
    ACCESS_FINE_LOCATION" />
```

```
05  <uses-permission android:name=  "android.permission.
    ACCESS COARSE LOCATION" />
06  <uses-permission android:name="android.permission.MODIFY PHONE STATE" />
07  <uses-permission android:name="android.permission.CHANGE WIFI STATE" />
08  <uses-permission android:name="android.permission.READ PHONE STATE" />
09  <uses-permission android:name="android.permission. WRITE EXTERNAL
    STORAGE" />
10  <uses-permission android:name="android.permission.MOUNT UNMOUNT
    FILESYSTEMS" />
```

到这里已经准备完成，可以在布局文件中加入 MapView 了，代码如下所示。最后写一个类继承 MapAcitivity，把布局文件设置进去就可以在地图上显示了。

```
1   <com.baidu.mapapi.MapView
2       android:id="@+id/mv_locate_map"
3       android:layout_width="fill_parent"
4       android:layout_height="fill_parent"
5       android:clickable="true" />
```

15.2　创建地图应用

接下去我们要根据地图的 API 实现一个简单的功能，输入城市名称和公交线路，查询该公交线路在地图上的分布情况。

15.2.1　新建布局文件

新建布局文件 busline.xml 如下所示，主要提供两个 EditText 供用户输入，一个是城市名称，默认是深圳，一个是公交线路名称，默认是 215 路。输入完成后单击“搜索”按钮，地图中将会显示搜索结果。

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <LinearLayout xmlns:android="http://schemas.android. com/apk/res/
   android"
03     android:orientation="vertical"
04     android:layout width="fill parent"
05     android:layout height="fill parent"
06     >
07     <!-- 用户交互区 -->
08     <LinearLayout
09         xmlns:android="http://schemas.android.com/apk/res/android"
10         android:orientation="horizontal"
11         android:layout width="fill parent"
12         android:layout height="wrap content">
13     <TextView
14         android:text="在"
15         android:layout_width="wrap_content"
16         android:layout_height="wrap_content" />
17         <!-- 城市名称 -->
18         <EditText
19             android:id="@+id/city"
20             android:layout width="wrap content"
21             android:layout height="wrap content"
22             android:text="深圳" />
```

```
23          <TextView
24              android:text="市内找"
25              android:layout width="wrap content"
26              android:layout height="wrap content" />
27          <!-- 线路名称 -->
28          <EditText
29              android:id="@+id/searchkey"
30              android:layout width="wrap content"
31              android:layout_height="wrap_content"
32              android:text="215" />
33          <TextView
34              android:layout_width="wrap_content"
35              android:layout_height="wrap_content"
36              android:text="路公交车" />
37          <Button
38              android:id="@+id/search"
39              android:text="搜索"
40              android:layout width="fill parent"
41              android:layout_height="wrap_content" />
42      </LinearLayout>
43      <!-- 地图界面 -->
44      <com.baidu.mapapi.MapView
45          android:id="@+id/bmapView"
46          android:layout width="fill parent"
47          android:layout height="fill parent"
48          android:clickable="true"
49      />
50  </LinearLayout>
```

15.2.2　新建程序管理类

新建程序管理类 BMapApp 继承于 Application，代码如下所示，将 mStrKey 的值设置为之前获得的 Key 值，在 onCreate()函数中初始化地图，并设置接听器接听初始化的结果，如代码 40 行所示。监听器 MyGeneralListener 实现 MKGeneralListener 的接口，用于监听初始化过程的出错事件，onGetNetworkState 和 onGetPermissionState 分别用于获取网络出错事件和 Key 授权失败事件。onTerminate 用于结束地图的连接，在退出程序前调用。

```
01  //程序管理类
02  public class BMapApp extends Application {
03      static BMapApp mDemoApp;
04      //百度 MapAPI 的管理类
05      BMapManager mBMapMan = null;
06
07      //授权 Key
08      //TODO：请输入您的 Key
09      //申请地址：http://dev.baidu.com/wiki/static/imap/key/
10      String mStrKey = "5288097969F6ABF9F9C2FAAE9ED0C8F438BC4AFD";
11      //授权 Key 正确，验证通过
12      boolean m bKeyRight = true;
13
14      //常用事件监听，用来处理通常的网络错误、授权验证错误等
15      static class MyGeneralListener implements MKGeneralListener {
16          @Override
17          public void onGetNetworkState(int iError) {
18            Log.d("MyGeneralListener", "onGetNetworkState error is"
```

```
            +iError);
19          Toast.makeText(BMapApp.mDemoApp.getApplicationContext(),
            "您的网络出错啦！",
20                  Toast.LENGTH LONG).show();
21      }
22      @Override
23      public void onGetPermissionState(int iError) {
24          Log.d("MyGeneralListener", "onGetPermissionState error is
            "+ iError);
25          if (iError ==  MKEvent.ERROR PERMISSION DENIED) {
26              //授权 Key 错误
27            Toast.makeText(BMapApp.mDemoApp. getApplicationContext(),
28                      "请在 BMapApiDemoApp.java 文件输入正确的授权 Key! ",
29                      Toast.LENGTH LONG).show();
30              BMapApp.mDemoApp.m bKeyRight = false;
31          }
32      }
33  }
34
35  @Override
36  public void onCreate() {
37      Log.v("BMapApiDemoApp", "onCreate");
38      mDemoApp = this;
39      mBMapMan = new BMapManager(this);
40      boolean isSuccess = mBMapMan. init(this.mStrKey, newMyGenera
        lListener());
41      //初始化地图 sdk 成功，设置定位监听时间
42      if (isSuccess) {
43          mBMapMan.getLocationManager().setNotifyInternal(10, 5);
44      }
45      else {
46          //地图 sdk 初始化失败，不能使用 sdk
47      }
48      super.onCreate();
49  }
50
51  @Override
52  //建议在您 app 的退出之前调用 mapadpi 的 destroy()函数,避免重复初始化带来的时间消耗
53  public void onTerminate() {
54      // TODO Auto-generated method stub
55      if (mBMapMan != null) {
56          mBMapMan.destroy();
57          mBMapMan = null;
58      }
59      super.onTerminate();
60  }
61 }
```

15.2.3　地图的主界面

地图的主界面实现类 BaiduDituActivity 如下所示，继承于 MapActivity，该类的大部分函数与 Activity 类似。首先执行 onCreate()函数初始化界面，将 busline 设置为当前界面，接着获得应用程序实例 app 来初始化地图管理器并调用函数 initMapActivity()来初始化地图 Activity，如代码 045、047 行所示。

初始化完地图之后还需要设置地图的一些控件及开始状态，如代码 051～062 行所示，设置地图带有放大缩小控件、设置地图当前的经纬度为深圳等等。最后初始化搜索模块并

注册事件监听器，在事件监听器中有很多接口函数，包括自驾路线、公交线路、走路路线等等，这里我们只用到公交线路，因此只需实现该接口函数即可。

当用户输入完城市信息和公交线路信息时，单击“搜索”按钮将调用函数SearchButtonProcess()查询结果，并将结果显示到地图上，如图 15.3 所示。

```
001 package guo.com.baiduditu;              //声明包语句
002~023 行为引入相关类，这里不再列举，请阅读光盘内容
//……
024 //地图主界面
025 public class BaiduDituActivity extends MapActivity {
026     //“搜索”按钮
027     Button mBtnSearch = null;
028     //地图 View
029     MapView mMapView = null;
030     //搜索模块，也可去掉地图模块独立使用
031     MKSearch mSearch = null;
032     //城市名称
033     String  mCityName = null;
034     protected void onCreate(Bundle savedInstanceState) {
035         super.onCreate(savedInstanceState);
036         setContentView(R.layout.busline);
037         //取得应用程序实例
038         BMapApp app = (BMapApp)this.getApplication();
039         if (app.mBMapMan == null) {
040             //实例化地图管理器
041             app.mBMapMan = new BMapManager(getApplication());
042             //初始化地图管理器
043             app.mBMapMan.init(app.mStrKey,new BMapApp.
                MyGeneralListener());
044         }
045         app.mBMapMan.start();
046         //初始化地图 Activity
047         super.initMapActivity(app.mBMapMan);
048         //获得地图界面元素
049         mMapView = (MapView)findViewById(R.id.bmapView);
050         //设置放大缩小控制器
051         mMapView.setBuiltInZoomControls(true);
052         //设置在缩放动画过程中也显示 overlay，默认为不绘制
053         mMapView.setDrawOverlayWhenZooming(true);
054         //得到 mMapView 的控制权，可以用它控制和驱动平移及缩放
055         MapController mMapController = mMapView.getController();
056         //用给定的经纬度构造一个 GeoPoint，单位是微度 (度 * 1E6)
057         GeoPoint point = new GeoPoint((int) (22.522 * 1E6),
058                 (int) (114.051 * 1E6));
059         //设置地图中心点
060         mMapController.setCenter(point);
061         //设置地图 zoom 级别
062         mMapController.setZoom(14);
063         //初始化搜索模块，注册事件监听
064         mSearch = new MKSearch();
```

```
065         mSearch.init(app.mBMapMan, new MKSearchListener(){
066             public void onGetPoiResult(MKPoiResult res, int type, interror){
067                 //错误号可参考 MKEvent 中的定义
068                 if (error != 0 || res == null) {
069                     Toast.makeText(BaiduDituActivity.this,
                        "抱歉，未找到结果", Toast.LENGTH LONG).show();
070                     return;
071                 }
072                 //找到公交路线 poi node
073                 MKPoiInfo curPoi = null;
074                 int totalPoiNum  = res.getNumPois();
075                 for( int idx = 0; idx < totalPoiNum; idx++ ) {
076                     curPoi = res.getPoi(idx);
077                     if ( 2 == curPoi.ePoiType ) {
078                       // poi 类型，0：普通点，1：公交站，2：公交线路，3：地铁站，
                          4：地铁线路
079                         mSearch.busLineSearch(mCityName, curPoi.uid);
080                         break;
081                     }
082                 }
083                 //没有找到公交信息
084                 if (curPoi == null) {
085                     Toast.makeText(BaiduDituActivity.this,
                        "抱歉，未找到结果", Toast.LENGTH_LONG).show();
086                     return;
087                 }
088             }
089             public void onGetDrivingRouteResult(MKDrivingRouteResult res,
090                     int error) {
091             }
092             public void onGetTransitRouteResult(MKTransitRouteResult res,
093                     int error) {
094             }
095             public void onGetWalkingRouteResult(MKWalkingRouteResult res,
096                     int error) {
097             }
098             public void onGetAddrResult(MKAddrInfo res, int error) {
099             }
100             public void onGetBusDetailResult(MKBusLineResult result,
            int iError) {
101                 if (iError != 0 || result == null) {
102                     Toast.makeText(BaiduDituActivity.this,
                    "抱歉，未找到结果", Toast.LENGTH_LONG).show();
103                     return;
104                 }
105                 RouteOverlay routeOverlay = new RouteOverlay
                (BaiduDituActivity.this, mMapView);
106                 //此处仅展示一个方案作为示例
107                 routeOverlay.setData(result.getBusRoute());
108                 mMapView.getOverlays().clear();
```

```
109                 //添加路线到原有地图上
110                 mMapView.getOverlays().add(routeOverlay);
111                 //更新地图
112                 mMapView.invalidate();
113                //设置动画，显示到路线的开始端
114                mMapView.getController().animateTo(result.getBusRoute()
               .getStart());
115             }
116             @Override
117             public void onGetSuggestionResult(MKSuggestionResult res,
                int arg1) {
118                 // TODO Auto-generated method stub
119             }
120             @Override
121             public void onGetRGCShareUrlResult(String arg0, int arg1) {
122                 // TODO Auto-generated method stub
123             }
124         });
125         //设定“搜索”按钮的响应
126         mBtnSearch = (Button)findViewById(R.id.search);
127         OnClickListener clickListener = new OnClickListener(){
128             public void onClick(View v) {
129                     SearchButtonProcess(v);
130             }
131         };
132         mBtnSearch.setOnClickListener(clickListener);
133     }
134     void SearchButtonProcess(View v) {
135         if (mBtnSearch.equals(v)) {
136             //取得城市信息
137             EditText editCity = (EditText)findViewById(R.id.city);
138             //取得公交线路信息
139             EditText editSearchKey = (EditText)findViewById(R.id.
                searchkey);
140             mCityName = editCity.getText().toString();
141             mSearch.poiSearchInCity(mCityName, editSearchKey.
               getText().toString());
142         }
143     }
144     //暂停
145     @Override
146     protected void onPause() {
147         BMapApp app = (BMapApp)this.getApplication();
148         app.mBMapMan.stop();
149         super.onPause();
150     }
151     //恢复
152     @Override
153     protected void onResume() {
154         BMapApp app = (BMapApp)this.getApplication();
```

```
155         app.mBMapMan.start();
156         super.onResume();
157     }
158     @Override
159     protected boolean isRouteDisplayed() {
160         // TODO Auto-generated method stub
161         return false;
162     }
163 }
```



图 15.3　公交线路搜索结果

15.3　知 识 拓 展

前面我们通过一个小小的例子讲述了百度地图 API 的基本使用方法，下面我们对其中的一些细节作一个更深入的讲解。

第一个地方是 setNoitifyInternal(maxTime,minTime)，这个函数的作用是什么呢？它主要用来设置地图的更新时间间隔，第一个值为最大间隔，第二个值为最小间隔，大家可以根据需要自己设置，一般设置 5～10 之间。

第二个地方是 mMapView.getOverlays().clear()，这个函数的作用是清除所有标记，之所以要清除标记，是因为我们设置了每隔几秒钟重绘地图标记，如果不清楚之前的标记，过一段时间，地图就布满了标记，对性能有影响。

第三个地方是设置地图缩放等级 mMapController.setZoom(14)，等级参数为 0～18 之间的整数，值越大比例尺越小，显示的内容越具体。

15.4　本 章 小 结

本章简要介绍了基于百度地图 API 的基本应用，要想在自己程序中嵌入地图应用需要熟练掌握百度地图的 API 函数，熟悉整个地图 API 的调用流程，才不会在使用过程中出现稀奇古怪的问题。

第 16 章　新浪微博客户端

如今微博已经越来越流行了，这应该归功于群众分享的意愿越来越强烈，大家希望通过微博分享自己所遇到的事、所看到的景、所接触的人等等。新浪微博作为中国最早的具有一定规模的微博，为广大群众提供了一个广阔的平台。现在新浪微博又提供了各种客户端，使用户可以通过各种各样的客户端及时地分享身边的事物。本章我们将介绍如何开发基于 Android 的新浪微博客户端。

16.1　开发前的准备

16.1.1　申请微博账号和获得授权

既然要开发新浪微博客户端，首先肯定是申请一个微博账号，申请了账号后就登录到新浪微博的开发平台。单击“我也要做开发者”按钮，如图 16.1 所示，进入之后选择创建一个移动应用，如图 16.2 所示。



图 16.1　进入开放平台

图 16.2　创建一个新应用

按照指定的要求填好表格提交，之后应用便创建成功了。进入“我的应用”，单击刚才创建的应用，查看应用的详细信息，如图 16.3 所示，可以看到 App Key 和 App Secret，这两个是我们等一下需要用到的。另外一点，创建第一个应用的时候，需要提交一些个人信息供审核，可能需要等一段时间才能看到 App Key 和 App Secret。在填写个人信息的时候，需要注意的一点是，要填写回调页，因为等下进行用户授权时需要用到。当然，回调页是随意填写的，也可以在“应用信息”→“高级信息”中进行设置。



图 16.3　应用基本信息

16.1.2　OAuth 认证介绍

大部分 API 的访问如发表微博、获取私信、关注都需要用户身份。目前新浪微博开放平台用户身份鉴权有 OAuth 2.0 和 Basic Auth（仅用于应用所属开发者调试接口）两种方式。OAuth 2.0 较 1.0 相比整个授权验证流程更简单更安全，也是未来最主要的用户身份验证和授权方式。

授权认证的流程如图 16.4 所示，其中 Client 指第三方应用，Resource Owner 指用户，Authorization Server 是我们的授权服务器，Resource Server 是 API 服务器。

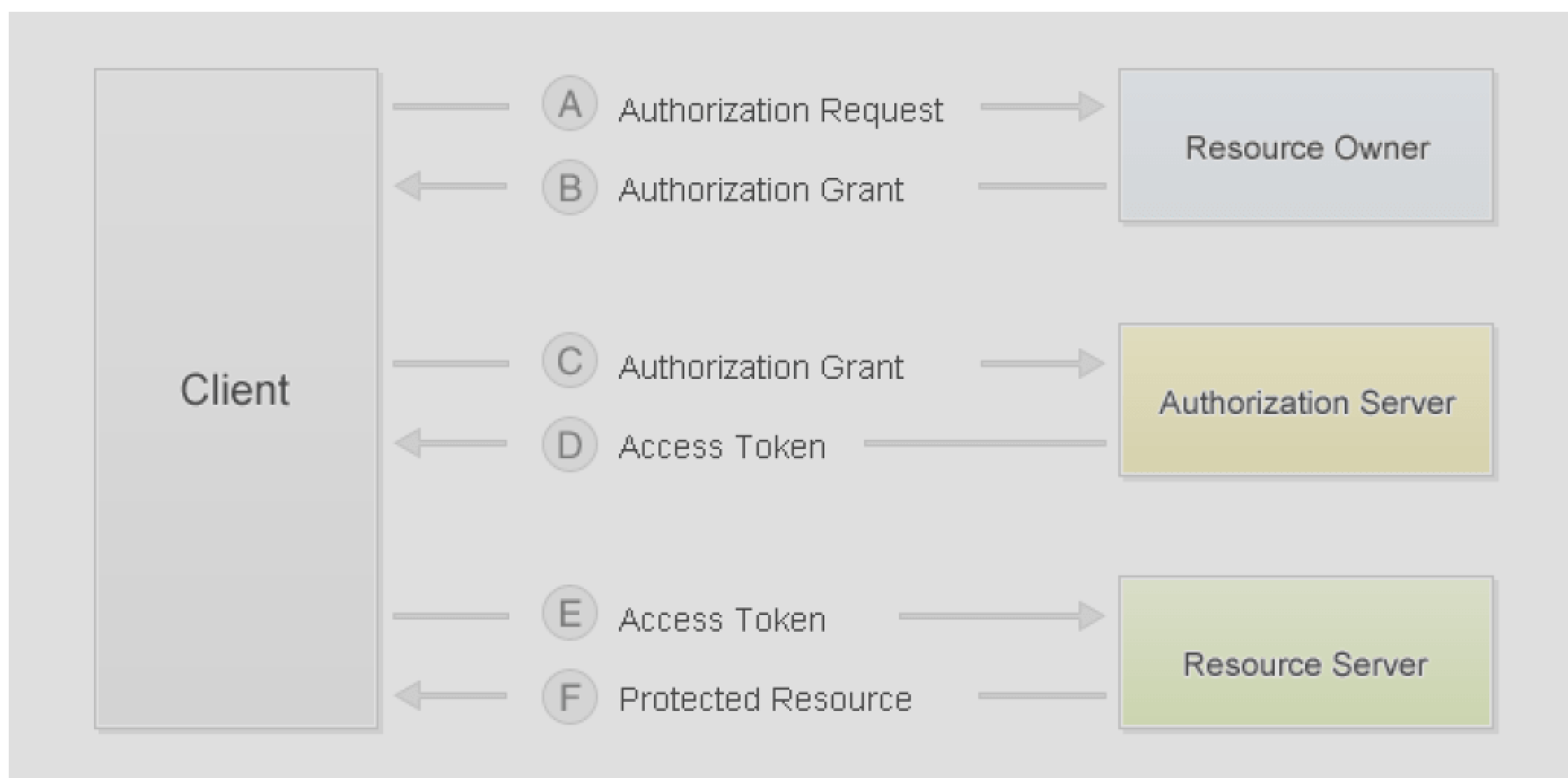


图 16.4　授权认证过程

授权的整个过程是这样的，首先需要引导用户到一个授权界面，引导过程需要用到前面我们申请的 App Key 和 App Secret 以及重定向地址。在这个界面中用户输入自己的账号和密码，然后单击授权来授权给应用程序访问自己账号的一些相关信息。之后用户请求认证服务器来获得 Access Token，有了这个 Access Token，你就可以访问资源服务器的一些内容，可以在用户授权范围内做任何事情了。

这个过程看似很简单，但我相信每个人实际去操作的时候会遇到各种问题，就像笔者。刚开始我一直不明白重定向地址的作用，想知道服务器是如何利用这个地址的，后来仔细看了文档，才明白这个地址只是用来返回参数用的，并没有实际的意义。还有这个 Access Token，其实只要有了这个 Access Token，你就可以做任何授权范围内的事了，但我也是绕了很大一个弯子才明白的，其实说到底还是对这种授权机制的理解不够导致的。其实这种机制很通用，很多授权机制无外乎这些套路，只要你彻底理解了一种，以后再碰到只要依葫芦画瓢即可。为了让读者少走弯路，接下去讲解登录过程的时候会跟大家分享我当时遇到的状况。

16.1.3　SDK 使用说明

（1）将 SDK 的工程项目导入到 Eclipse 中。

在 Eclipse 中单击 File|Import|General|Existing Projects into Workspace 命令。注意：SDK 工程的编码格式为 UTF-8，如图 16.5 所示。

（2）在需要集成 SDK 的工程项目中添加 Library。

右键单击|Properties |Android 命令，设置 Library 属性，如图 16.6 所示。

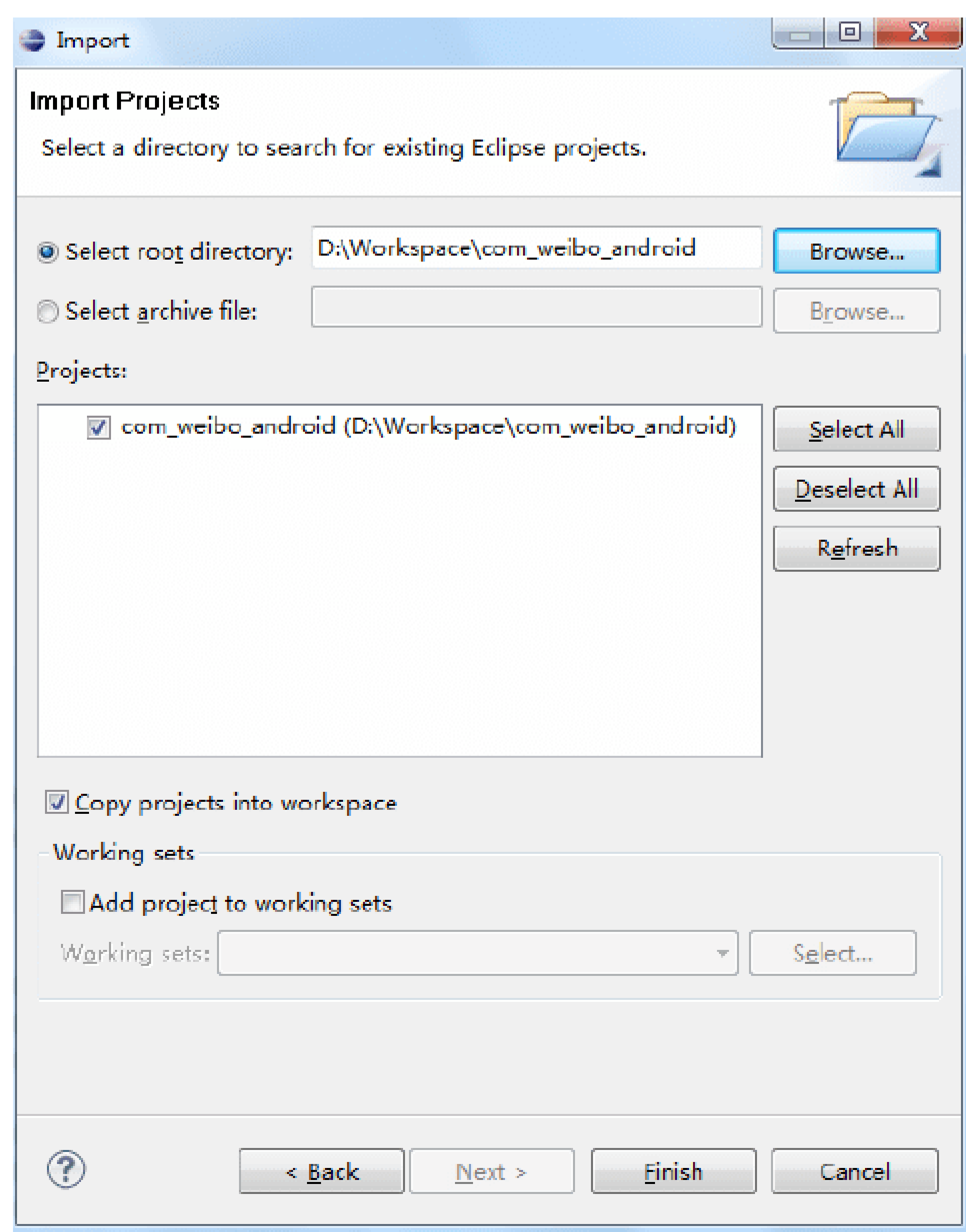


图 16.5　将 SDK 工程导入 Eclipse

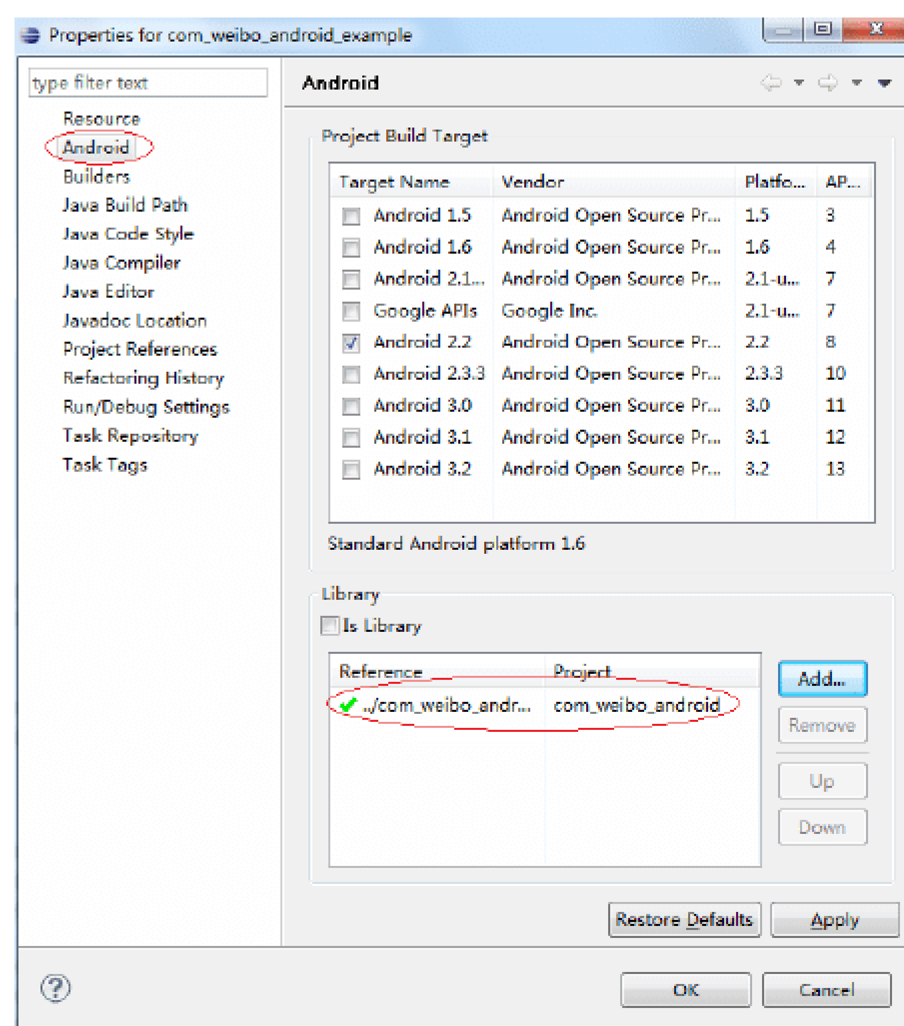


图 16.6　导入 Libaray

（3）Manifest 文件中必须包含以下 permission：

```
1  <uses-permission android:name="android.permission.INTERNET">
   </uses-permission>
2  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE">
   </uses-permission>
3  <uses-permission android:name="android.permission.
    WRITE_APN_SETTINGS"></uses-permission>
4  <uses-permission android:name="android.permission.
    CHANGE_WIFI_STATE"></uses-permission>
```

代码中将 APP_KEY、APP_SECRET 存放在 Weibo 类中，在 Weibo.java 中设置 APP_KEY 和 APP_SECRET，如下所示：

```
1   private static String APP KEY = "2081498321";
2   private static String APP_SECRET = "7907ee613510bb7d3b2ce8beed302dd1";
```

16.2　载入界面设计

考虑到本程序只作为一个 demo，并没有太考究界面的设计，读者在明白了整个设计的过程之后，可以根据喜好设计 UI。

首先，登录 UI，简单地 PS 了一下，将几张图拼在一起，如图 16.7 所示。



图 16.7　载入 UI 界面

如下所示，在 main.xml 中设置背景为该图片即可。

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout xmlns:android="http://schemas.android. com/apk/res/
     android"
3      android:layout width="fill parent"
4      android:layout_height="fill_parent"
5      android:background="@drawable/main ui"
6      android:orientation="vertical" >
```

16.3　载入界面功能实现

在载入界面出现的时候，我们在后台做什么呢？是的，我们需要载入数据库中的用户信息，这里我们使用 sqlite 作为数据存储的方式。因此首先我们需要新建一个数据表用于存储用户信息，并设置数据表的读取方式，与此同时在数据库与外界进行数据存取的时候，需要一个类用于保存用户信息。

16.3.1　保存用户信息

首先，新建一个类 UserInfo.class 用于保存用户的信息，代码如下所示，前面定义的静态字符串变量为用户数据库中对应的字段名，我们今后在进行数据库相关操作需要用到这些字段的时候直接引用就行。接下去定义了 6 个字段，分别是用户数据库的 id、用户微博 id、用户微博昵称、用户的 Access_Token、用户的 App_Secret 和用户的图标。

```
01  package com.guo.weibo;
02
03  import android.graphics.drawable.Drawable;
04
05  public class UserInfo {
06
07      final static String ID = "ID";
08      final static String USERID = "USERID";
09      final static String USERNAME = "USERNAME";
10      final static String TOKEN = "TOKEN";
11      final static String TOKENSECRET = "TOKENSECRET";
12      final static String USERICON = "USERICON";
13
14      //用户在数据库中的 id，递增
15      private String id;
16      //用户微博 id
17      private String userId;
18      //用户微博昵称
19      private String userName;
20      //用户 Access Token，为经过授权后的 Token，用于读取用户数据等操作
21      private String token;
22      //App 的 App_Secret
23      private String tokenSecret;
24      //用户图标
25      private Drawable userIcon;
26      //设置 id
27      public void setId(String id)
28      {
29          this.id=id;
30      }
31      //设置用户微博 id
32      public void setUserId(String userId)
33      {
34          this.userId=userId;
35      }
36      //设置用户微博昵称
37      public void setUserName(String userName)
38      {
39          this.userName=userName;
40      }
41      //设置 Access_Token
42      public void setToken(String token)
43      {
44          this.token=token;
45      }
46      //设置 App_Secret
47      public void setTokenSecret(String tokenSecret)
48      {
```

```
49          this.tokenSecret=tokenSecret;
50      }
51      //设置用户图标
52      public void setUserIcon(Drawable userIcon)
53      {
54          this.userIcon=userIcon;
55      }
56      //取得 id
57      public String getId()
58      {
59          return id;
60      }
61      //取得用户微博 id
62      public String getUserId()
63      {
64          return userId;
65      }
66      //取得微博用户名
67      public String getUserName()
68      {
69          return userName;
70      }
71      //取得 Acces_Token
72      public String getToken()
73      {
74          return token;
75      }
76      //取得 App Secret
77      public String getTokenSecret()
78      {
79          return tokenSecret;
80      }
81      //取得用户图标
82      public Drawable getUserIcon()
83      {
84          return userIcon;
85      }
```

16.3.2　新建数据库

接着新建一个数据库操作类，而为了方便对数据库进行版本管理，我们使用的类继承于 SQLiteOpenHelper 类，它提供了两个重要的方法，分别是 onCreate(SQLiteDatabase db)和 onUpgrade(SQLiteDatabase db,int oldVersion,int vewVersion)，前者用于初次使用这个类时生成数据库，后者用于更新数据库表结构。第一次调用这个类的时候将会执行 onCreate 语句，之后只要数据库存在则不再执行 onCreate。最下面的 updateColunm 方法用于对数据表的列进行更新。

```
01  package com.guo.weibo;                //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容
//……
08  public class SqliteHelper extends SQLiteOpenHelper{
09
10      //用来保存 UserID、Access Token、Access Secret 的表名
11      public static final String TB_NAME="users";
```

```
12      public SqliteHelper(Context context, String name,
        CursorFactory factory, int version) {
13      super(context, name, factory, version);
14      }
15      //创建表
16      @Override
17      public void onCreate(SQLiteDatabase db) {
18          db.execSQL("CREATE TABLE IF NOT EXISTS "+
19          TB_NAME+"("+
20          UserInfo.ID+" integer primary key,"+
21          UserInfo.USERID+" varchar,"+
22          UserInfo.TOKEN+" varchar,"+
23          UserInfo.TOKENSECRET+" varchar,"+
24          UserInfo.USERNAME+" varchar,"+
25          UserInfo.USERICON+" blob"+
26          ")"
27          );
28          Log.e("Database","onCreate");
29      }
30      //更新表
31      @Override
32      public void onUpgrade(SQLiteDatabase db, int oldVersion,
        int newVersion) {
33          db.execSQL("DROP TABLE IF EXISTS " + TB_NAME);
34          onCreate(db);
35          Log.e("Database","onUpgrade");
36      }
37      //更新列
38      public void updateColumn(SQLiteDatabase db, String oldColumn,
        String newColumn, String typeColumn){
39          try{
40          db.execSQL("ALTER TABLE " +
41          TB_NAME + " CHANGE " +
42          oldColumn + " "+ newColumn +
43          " " + typeColumn
44          );
45          }catch(Exception e){
46          e.printStackTrace();
47          }
48      }
```

16.3.3 “增删改查”数据

接下来新建一个类 DataHelper 用于操作数据表的数据，即对数据进行“增删改查”。GetUserList 用于获取数据库中所有用户的信息，参数 isSample 为 false 时返回的用户数据带有昵称和图标，而 isSample 为 true 时不带用户昵称和图标。

GetUserByName 函数传入用户的昵称可以获得用户的数据，SaveUseInfo 和 DelUserInfo 分别用于保存和删除用户数据。UpdateUserInfo 有两个同名函数，根据参数不同可以选择只更新基本信息：Userid、Token、TokenSecret 或者全部信息。

```
001 package com.guo.weibo;                  //声明包语句
002~015 行为引入相关类，这里不再列举，请阅读光盘内容
//……
016 public class DataHelper {
017     //数据库名称
```

```
018     private static String DB_NAME = "sinaweibo.db";
019     //数据库版本
020     private static int DB VERSION = 2;
021     private SQLiteDatabase db;
022     private SqliteHelper dbHelper;
023
024     public DataHelper(Context context){
025         dbHelper=new SqliteHelper(context,DB_NAME, null, DB_VERSION);
026         db= dbHelper.getWritableDatabase();
027     }
028
029     public void Close()
030     {
031         db.close();
032         dbHelper.close();
033     }
034     //获取 users 表中的 UserID、Access Token、Access Secret 的记录
035     public List<UserInfo> GetUserList(Boolean isSimple)
036     {
037         List<UserInfo> userList = new ArrayList<UserInfo>();
038         Cursor cursor=db.query(SqliteHelper.TB_NAME, null, null, null,
            null, null, UserInfo.ID+" DESC");
039         cursor.moveToFirst();
040         while(!cursor.isAfterLast()&& (cursor.getString(1)!=null)){
041         UserInfo user=new UserInfo();
042         user.setId(cursor.getString(0));
043         user.setUserId(cursor.getString(1));
044         user.setToken(cursor.getString(2));
045         user.setTokenSecret(cursor.getString(3));
046     if(!isSimple){
047         if(cursor.getString(4) != null && cursor.getBlob(5) != null)
048         {
049             user.setUserName(cursor.getString(4));
050             ByteArrayInputStream stream = newByteArrayInputStream
                (cursor.getBlob(5));
051             Drawable icon= Drawable.createFromStream(stream, "image");
052             user.setUserIcon(icon);
053         }
054     }
055         userList.add(user);
056         cursor.moveToNext();
057     }
058         cursor.close();
059         return userList;
060     }
061
062     //判断 users 表中是否包含某个 UserID 的记录
063     public Boolean HaveUserInfo(String UserId)
064     {
065         Boolean b=false;
066         Cursor cursor=db.query(SqliteHelper.TB NAME, null,
            UserInfo.USERID + "=" + UserId, null, null, null,null);
067         b=cursor.moveToFirst();
068         Log.e("HaveUserInfo",b.toString());
069         cursor.close();
070         return b;
071     }
072     //判断 users 表中是否包含某个 UserID 的记录
073     public UserInfo GetUserByName(String userName)
074     {
075         Boolean b=false;
```

```
076         //注意汉字为查询条件时需要加''
077         Cursor cursor=db.query(SqliteHelper.TB_NAME, null, UserInfo.
            USERNAME + "='" + userName+"'", null, null, null,null);
078         b=cursor.moveToFirst();
079         Log.e("GetUserByName",b.toString());
080         if(b != false){
081             UserInfo user=new UserInfo();
082             user.setId(cursor.getString(0));
083             user.setUserId(cursor.getString(1));
084             user.setToken(cursor.getString(2));
085             user.setTokenSecret(cursor.getString(3));
086             user.setUserName(cursor.getString(4));
087             ByteArrayInputStream stream = newByteArrayInputStream
                (cursor.getBlob(5));
088             Drawable icon= Drawable.createFromStream(stream, "image");
089             user.setUserIcon(icon);
090             cursor.close();
091             return user;
092         }
093         return null;
094     }
095     //更新 users 表的记录，根据 UserId 更新用户昵称和用户图标
096     public int UpdateUserInfo(String userName,Bitmap userIcon,String
        UserId)
097     {
098         ContentValues values = new ContentValues();
099         values.put(UserInfo.USERNAME, userName);
100         // BLOB 类型
101         final ByteArrayOutputStream os = new ByteArrayOutputStream();
102         //将 Bitmap 压缩成 PNG 编码，质量为 100%存储
103         userIcon.compress(Bitmap.CompressFormat.PNG, 100, os);
104         //构造 SQLite 的 Content 对象，这里也可以使用 raw
105         values.put(UserInfo.USERICON, os.toByteArray());
106         int id= db.update(SqliteHelper.TB_NAME, values, UserInfo.
            USERID + "=" + UserId, null);
107         Log.e("UpdateUserInfo2",id+"");
108         return id;
109     }
110
111     //更新 users 表的记录
112     public int UpdateUserInfo(UserInfo user)
113     {
114         ContentValues values = new ContentValues();
115         values.put(UserInfo.USERID, user.getUserId());
116         values.put(UserInfo.TOKEN, user.getToken());
117         values.put(UserInfo.TOKENSECRET, user.getTokenSecret());
118         int id= db.update(SqliteHelper.TB_NAME, values, UserInfo.
            USERID + "=" + user.getUserId(), null);
119         Log.e("UpdateUserInfo",id+"");
120         return id;
121     }
122
123     //添加 users 表的记录
124     public Long SaveUserInfo(UserInfo user)
125     {
126         ContentValues values = new ContentValues();
127         values.put(UserInfo.USERID, user.getUserId());
128         values.put(UserInfo.TOKEN, user.getToken());
129         values.put(UserInfo.TOKENSECRET, user.getTokenSecret());
130         Long uid = db.insert(SqliteHelper.TB_NAME, UserInfo.ID, values);
```

```
131         Log.e("SaveUserInfo",uid+"");
132         return uid;
133     }
134
135     //删除 users 表的记录
136     public int DelUserInfo(String UserId){
137         int id= db.delete(SqliteHelper.TB_NAME, UserInfo.
            USERID +"="+UserId, null);
138         Log.e("DelUserInfo",id+"");
139         return id;
140     }
141 }
```

16.3.4　获取数据库所有的信息

接着在 MainActivity 的 onCreate()函数中获取用户数据库中的所有信息，判断数据是否为空。如果为空，说明还没有授权过，弹出对话框提示用户是否进行授权，单击“确定”按钮可以进入授权页面进行授权，否则退出本程序。相反，如果用户数据库中有数据，则直接进入登录界面。这里需要注意的一点是，在进入其他 Activity 的时候需要用 finish()关闭本 Activity。

为了能看到载入界面的效果，我们在这里采用了延时处理。Android 延时执行某个任务有以下几种方法：

- ❑ 开启新线程。

```
1   new Thread(new Runnable(){
2       public void run(){
3           Thread.sleep(XXXX);
4           handler.sendMessage();//告诉主线程执行任务
5       }
```

- ❑ 利用定时器。

```
1   TimerTask task = new TimerTask(){
2       public void run(){
3       //execute the task
4       }
5   };
6   Timer timer = new Timer();
```

或者

```
1   new Handler().postDelayed(new Runnable(){
2       public void run() {
3       //execute the task
4       }
```

- ❑ 利用 AlarmManager。其实 AlarmManager 也是一种定时器，只不过它是全局的。

```
01    //操作：发送一个广播，广播接收后 Toast 提示定时操作完成
02    Intent intent =new Intent(Main.this, alarmreceiver.class);
03    intent.setAction("short");
04    PendingIntent sender=
05    PendingIntent.getBroadcast(Main.this, 0, intent, 0);
06
07    //设定一个 5 秒后的时间
```

```
08    Calendar calendar=Calendar.getInstance();
09    calendar.setTimeInMillis(System.currentTimeMillis());
10    calendar.add(Calendar.SECOND, 5);
11
12    AlarmManager alarm=(AlarmManager)getSystemService(ALARM_SERVICE);
13    alarm.set(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
      ender);
14    //或者以下面的方式简化
15    //alarm.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis
      ()+5*1000, sender);
16
17   Toast.makeText(Main.this, "五秒后 alarm 开启",
     Toast.LENGTH_LONG).show();
```

上面 3 种延迟执行任务的方式各有优势，其中第二种使用最简单，适合于简单应用的场合。第三种最复杂，但功能也最强大。

```
01  package com.guo.weibo;                //声明包语句
02~14 行为引入相关类，这里不再列举，请阅读光盘内容
//……
15  public class MainActivity extends Activity {
16      /** Called when the activity is first created. */
17      DataHelper dbHelper;
18      @Override
19      public void onCreate(Bundle savedInstanceState) {
20          super.onCreate(savedInstanceState);
21          setContentView(R.layout.main);
22          //获取账号列表
23          dbHelper=new DataHelper(this);
24          //获取数据库用户列表
25          final List<UserInfo> userList= dbHelper.GetUserList(true);
26          //关闭数据库
27          dbHelper.Close();
28          //为了看到载入界面的效果，我们延迟 1s 执行判断
29          TimerTask task=new TimerTask(){
30                  public void run()
31                  {
32                          //如果为空说明第一次使用，弹出对话框引导用户进行授权
33                          if(userList.isEmpty())
34                            {
35                              Builder mBuilder=new AlertDialog.Builder(Main
                                Activity.this);
36                              mBuilder.setTitle("提示");
37                              mBuilder.setMessage("您还未创建任何账户，是否现在
                                创建？");
38                              mBuilder.setPositiveButton("确定",
                                new OnClickListener(){
39                                  @Override
40                                  public void onClick(DialogInterface dialog,
                                    int which) {
41                                      // TODO Auto-generated method stub
42                                      Intent intent = new Intent();
43                                      //跳到AuthorizeActivity页面进行OAuth认证
44                                      intent.setClass(MainActivity.this,
                                       AuthorizeActivity.class);
45                                      startActivity(intent);
46                                      //关闭当前界面
47                                      MainActivity.this.finish();
```

```
48                              }
49                          }).setNegativeButton("取消", new
                            OnClickListener(){
50                              @Override
51                              public void onClick(DialogInterface dialog,
                                int which) {
52                                  // TODO Auto-generated method stub
53                                  //取消则关闭本程序
54                                  MainActivity.this.finish();
55                              }
56                          });
57                          mBuilder.create().show();
58                      }
59                      else
60                      {
61                          Intent it=new Intent();
62                          //如果不为空，则跳转到登录界面
63                          it.setClass(MainActivity.this,
                            LoginActivity.class);
64                          startActivity(it);
65                          //关闭本程序
66                          MainActivity.this.finish();
67                      }
68              }};
69          Timer timer=new Timer();
70          timer.schedule(task,1000);
71      }
```

16.4　授权功能实现

由于授权引导页我们简单地用了一个 AlertDialog 进行实现，如图 16.8 所示，因此这里关于授权页面的设计就略去了，下面我们直接来看一下授权功能的实现。

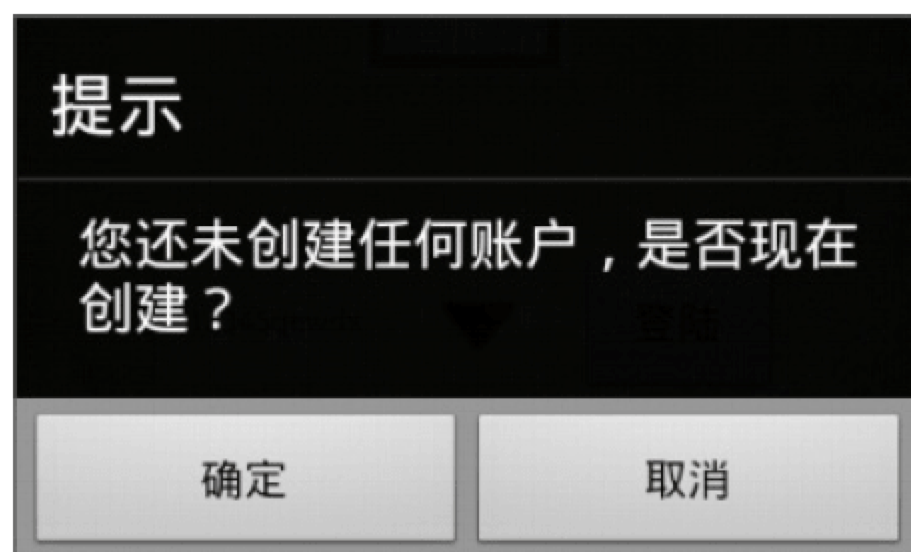


图 16.8　引导用户到授权页面

如下代码所示，我们首先声明一些将要用到的变量，比如 userInfo 用于保存当前授权成功后的用户信息，包括 Token、nickname 和 icon 等。接着在 onCreate()函数中实例化数据库、weibo 类，设置之前我们在 sina 微博注册得到的 3 个参数：App_Key、App_Secret 和 RedirectUrl。最后调用 authorize 函数进入认证界面，并设置认证结果监听器 AuthDialogListener。

之后用户会被引导到授权界面如图 16.9 所示，当用户输入正确的用户名和密码后，单击“授权”按钮就会向服务器发出授权请求，并返回我们的回调地址加上 access_token、

expire_in 等参数。我们在认证结果监听器 AuthDialogListener 中对返回的结果进行处理，使用 getString 方法分别获取返回的数据，如代码 055～058 行所示。然后新建一个 AccessToken 变量，并调用 weibo 的 setAccessToken 保存 Access_Token 信息，由于 weibo 类中保存的都是静态变量，因此当我们第一次实例化保存完这些数据之后，下一次再实例化这些数据仍然存在。

然后将相应的信息保存到 userInfo 这个变量中，并根据用户的 id 判断数据库中是否已有该用户的数据，如果有就调用 UpdateUserInfo，否则调用 SaveUserInfo。因为此时通过回调地址并没有获得用户的昵称和图标信息，因此我们需要利用刚才获得的 Access_Token 和 uid 来调用相应的 API 接口获取用户的详细信息。

函数 getUserDetail()用来获取用户的详细信息，需要传入 source 和 uid 作为参数，然后调用 weibo 的函数 request 获得服务器返回数据。为了方便读取，我们将获得的字符串转换成 JSON 对象。代码 105～123 行将获得的昵称和图标信息更新到数据库对应的 uid 中。

当数据存储完毕后，程序将调用 startActivity 跳转到登录界面，并关闭当前界面。

```
001 package com.guo.weibo;                      //声明包语句
002~017 行为引入相关类，这里不再列举，请阅读光盘内容
//……
018 //用户授权认证
019 public class AuthorizeActivity extends Activity{
020     DataHelper dbHelper;
021     //保存当前授权成功的用户信息
022     UserInfo userInfo;
023     //保存当前用户的 Access_Token
024     String token;
025     //用户保存当前授权成功的用户 id
026     String user id;
027     //实例化 weibo 类
028     Weibo weibo;
029     //保存当前的上下文
030     Context mContext;
031     @Override
032     public void onCreate(Bundle savedInstanceState)
033     {
034         super.onCreate(savedInstanceState);
035         mContext=this;
036         dbHelper=new DataHelper(this);
037         //实例化 weibo
038         weibo = Weibo.getInstance();
039         userInfo=new UserInfo();
040         //设置 App_Key 和 App_Secret
041         weibo.setupConsumerConfig(ConstParam.CONSUMER_KEY,
            ConstParam.CONSUMER_SECRET);
042         //Oauth 2.0
043         //隐式授权认证方式
044         //此处回调页内容应该替换为与 appkey 对应的应用回调页
045         weibo.setRedirectUrl(ConstParam.CALLBACK_URL);
046         //对应的应用回调页可在开发者登录新浪微博开发平台之后，
047         //进入我的应用→应用详情→应用信息→高级信息→授权设置→
              应用回调页进行设置和查看，
048         //应用回调页不可为空
049         weibo.authorize(this,new AuthDialogListener());
050     }
```

```
051       class AuthDialogListener implements WeiboDialogListener {
052             @Override
053             public void onComplete(Bundle values) {
054                 //获得 Access_Token
055                 token = values.getString("access_token");
056                 //获得过期时间
057                 String expires in = values.getString("expires in");
058                 user id = values.getString("uid");
059                 Log.e("sinaweibo","access token : " + token + "
                    expires in: "
060                         + expires in+" uid :"+user id);
061                 AccessToken accessToken = new AccessToken(token,
                    ConstParam.CONSUMER SECRET);
062                 accessToken.setExpiresIn(expires_in);
063                 //设置 Access_Token
064                 weibo.setAccessToken(accessToken);
065                 //保存用户信息
066                 userInfo.setToken(token);
067                 userInfo.setUserId(user id);
068                 userInfo.setTokenSecret(ConstParam.CONSUMER SECRET);
069                 if(dbHelper.HaveUserInfo(user id))
070                 {
071                     //更新用户信息
072                     dbHelper.UpdateUserInfo(userInfo);
073                 }else{
074                     //添加用户信息
075                     dbHelper.SaveUserInfo(userInfo);
076                 }
077                 new Thread(mRunnable).start();
078                 //进入登录界面
079                 Intent it=new Intent();
080                 it.setClass(AuthorizeActivity.this,
                    LoginActivity.class);
081                 startActivity(it);
082             }
083             //授权异常
084             @Override
085             public void onWeiboException(WeiboException e) {
086                 // TODO Auto-generated method stub
087                 Toast.makeText(AuthorizeActivity.this, e.getMessage(),
                    Toast.LENGTH LONG).show();
088                 AuthorizeActivity.this.finish();
089             }
090             //授权出错
091             @Override
092             public void onError(DialogError e) {
093                 // TODO Auto-generated method stub
094                 Toast.makeText(AuthorizeActivity.this, e.getMessage(),
                    Toast.LENGTH LONG).show();
095                 AuthorizeActivity.this.finish();
096             }
097             //授权被取消
098             @Override
099             public void onCancel() {
100                 // TODO Auto-generated method stub
101                 AuthorizeActivity.this.finish();
102             }
103         }
104       //获得昵称和用户图标
```

```
105         Runnable mRunnable=new Runnable(){
106           @Override
107           public void run() {
108               // TODO Auto-generated method stub
109               try {
110                   JSONObject json=getUserDetail(mContext,weibo,user id);
111                   //获得用户昵称
112                   String nickName = json.getString("screen_name");
113                   Log.e("guojs","nickname:"+nickName);
114                   Bitmap bm=Common.getBm
                      (new URL(json.getString("profile_image_url")));
115                   //更新用户信息
116                   dbHelper.UpdateUserInfo(nickName,bm,user id);
117                   //关闭数据库
118                   dbHelper.Close();
119               } catch (Exception e) {
120                   Log.e("guojs",e.getMessage());
121               }
122           }
123         };
124         //调用 API，获得用户信息
125         public JSONObject getUserDetail(Context mContext,
            Weibo weibo,String uid)
126         {
127             //获取用户信息 API
128             String url=Weibo.SERVER+"users/show.json";
129             WeiboParameters bundle=new WeiboParameters();
130             bundle.add("source", ConstParam.CONSUMER KEY);
131              //添加用户 id 作为参数
132             bundle.add("uid",uid);
133             String rlt="";
134               try {
135                   rlt = weibo.request(mContext, url, bundle, "GET", weibo.
                      getAccessToken());
136                   return new JSONObject(rlt);
137               } catch (Exception e) {
138                   // TODO Auto-generated catch block
139                   e.printStackTrace();
140                   return null;
141               }
142         }
```



图 16.9　用户授权界面

另外，上面用到了 Common 类，代码如下所示，login_user 用于保存当前登录成功的用户信息，weibo 用于保存用户授权信息等。getBm()函数通过传入一个图片的 url 地址来下载图片。

```
01  package com.guo.weibo;                        //声明包语句
02~10 行为引入相关类，这里不再列举，请阅读光盘内容
//……
11  //保存程序通用的函数和变量
12  public class Common {
13      public static UserInfo login_user=null;
14      public static Weibo weibo=Weibo.getInstance();
15      //通过 url 地址取得图像
16      public static Bitmap getBm(URL aURL)
17      {
18          URLConnection conn;
19          Bitmap bm=null;
20          try {
21              //打开 url 链接
22              conn = aURL.openConnection();
23              conn.connect();
24              //将数据流保存到 is
25              InputStream is = conn.getInputStream();
26              BufferedInputStream bis = new BufferedInputStream(is);
27              //用位图工厂解码数据流，将数据流转换成位图
28              bm = BitmapFactory.decodeStream(bis);
29              bis.close();
30              is.close();
31          } catch (IOException e) {
32              // TODO Auto-generated catch block
33              e.printStackTrace();
34          }
35          //返回位图
36          return bm;
37      }
```

16.5　登录界面设计

我们登录界面的最终实现效果如图 16.10 所示。

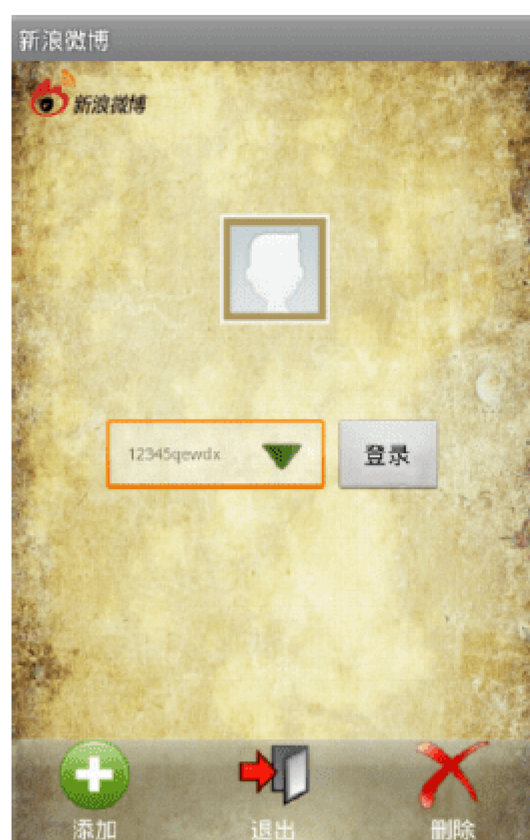


图 16.10　用户登录界面

16.5.1　设置布局结构

为了实现这种效果，我们在 res/layout 下新建 login.xml，采用 RelativeLayout 的布局结构，代码如下所示。使用一个 RelativeLayout 容器用来存放用户图像，并设置背景为 icon_bg。中间显示当前用户的昵称，单击旁边的绿色小三角可以显示当前授权过的所有用户信息。右边的“登录”按钮，顾名思义就是用来登录的。最下面一排使用一个 GridView 实现，具体的布局在程序的代码中实现。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout xmlns:android="http://schemas.android.
    com/apk/res/android"
03      android:layout_width="fill_parent"
04      android:layout_height="fill_parent"
05      android:background="@drawable/background" >
06      <!-- 用于显示当前用户 icon 的容器 -->
07          <RelativeLayout
08              android:id="@+id/iconBg"
09              android:layout_width="100px"
10              android:layout_height="100px"
11              android:background="@drawable/icon_bg"
12              android:layout_above="@+id/selectLayout"
13              android:layout_centerHorizontal="true"
14              android:layout_marginTop="140px" >
15              <!-- 显示用于 Icon -->
16              <ImageView
17                  android:id="@+id/icon"
18                  android:layout_width="80px"
19                  android:layout_height="80px"
20                  android:layout_centerInParent="true" />
21          </RelativeLayout>
22          <RelativeLayout
23              android:layout_width="fill_parent"
24              android:layout_height="fill_parent">
25              <RelativeLayout
26                  android:id="@+id/selectLayout"
27                  android:layout_width="wrap_content"
28                  android:layout_height="wrap_content"
29                  android:layout_centerInParent="true">
30                  <!-- 显示当前用户的昵称 -->
31                  <EditText
32                      android:id="@+id/iconSelect"
33                      android:layout_width="200px"
34                      android:layout_height="wrap_content"
35                      android:maxLength="10"
36                      android:paddingLeft="20px"
37                      android:editable="false"
38                      android:enabled="false"
39                      android:textSize="15px" />
40                  <!-- 显示所有用户按钮 -->
41                  <ImageButton
42                      android:id="@+id/iconSelectBtn"
43                      android:layout_width="wrap_content"
44                      android:layout_height="wrap_content"
45                      android:layout_alignBottom="@+id/iconSelect"
46                      android:layout_alignRight="@+id/iconSelect"
47                      android:layout_alignTop="@+id/iconSelect"
```

```
48                  android:layout marginRight="1.0dip"
49                  android:background="@drawable/down" />
50              <!-- 登录按钮 -->
51              <Button
52                  android:id="@+id/login"
53                  android:layout_width="100px"
54                  android:layout_height="40px"
55                  android:layout_marginLeft="5dip"
56                  android:layout_alignTop="@+id/iconSelectBtn"
57                  android:layout_toRightOf="@+id/iconSelectBtn"
58                  android:layout_alignBottom="@+id/iconSelectBtn"
59                  android:text="登录" />
60          </RelativeLayout>
61       <!-- 用于显示底部菜单栏 -->
62          <GridView
63              android:id="@+id/gridview_toolbar"
64              android:layout_height="wrap_content"
65              android:layout_width="fill_parent"
66              android:layout alignParentBottom="true"></GridView>
67      </RelativeLayout>
```

16.5.2　显示所有用户列表

当单击旁边的绿色按钮时将弹出当前所有授权用户的列表，如图 16.11 所示。



图 16.11　显示所有用户列表

这个弹出列表采用的是 ListView 界面元素，在 res/layout 下新建 user_list.xml，实现代码如下所示：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.
    com/apk/res/android"
03      android:id="@+id/layout_myview"
04      android:layout width="fill parent"
05      android:layout height="fill parent"
06      android:orientation="vertical" >
07      <!-- 用于显示所有用户的列表 -->
08      <ListView
09          android:id="@+id/userList"
10          android:layout_width="fill_parent"
11          android:layout_height="fill_parent"
12          android:divider="#ffffff"
```

```
13          android:dividerHeight="1dip"
14          android:listSelector="#BB000000" >
15      </ListView>
```

同时，为该 ListView 适配的每一行元素布局如下所示：

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout xmlns:android="http://schemas.android.
    com/apk/res/android"
03      android:id="@+id/RelativeLayout Item"
04      android:layout width="fill parent"
05      android:layout height="wrap content"
06      android:paddingBottom="5dip">
07      <!-- 用于显示菜单的图片 -->
08      <ImageView
09          android:id="@+id/item_image"
10          android:layout_centerHorizontal="true"
11          android:layout_width="wrap_content"
12          android:layout_height="45dp"></ImageView>
13      <!-- 用于显示菜单的文字 -->
14      <TextView
15          android:layout below="@id/item image"
16          android:id="@+id/item text"
17          android:layout centerHorizontal="true"
18          android:layout_width="wrap_content"
19          android:layout_height="wrap_content"
20          android:textColor="#FFFFFFFF"></TextView>
```

16.6　登录界面功能实现

如下代码所示，声明了一些成员变量：

```
01  public class LoginActivity extends Activity{
02  //数据库操作类
03  DataHelper dbHelper;
04  //用户信息数据适配器
05  UserAdapter mUserAdapter;
06  //用于保存数据库中所有用户的信息
07  List<UserInfo> userList;
08  //用于显示当前用户的昵称
09  EditText username;
10  //保存当前的上下文
11  Context mContext;
12  //当前选中的用户信息
13  UserInfo selectedUser;
14  Weibo weibo;
15  //当前选中的用户 id
16  String user_id;
17  //当前选中用户的图标
18  ImageView icon;
19  //底部菜单栏
20  GridView mGridView;
21  //底部菜单对应的标题
22  String[] gridview_menu_title={"添加","退出","删除"};
23  AlertDialog alertDialog;
24  //底部菜单对应的图标
```

```
25  int[] girdview menu image={R.drawable.add,R.drawable.
    exit,R.drawable.del};
```

16.6.1　初始化界面图标

在 onCreate()函数中，首先实例化数据库，初始化界面图标，并为按键绑定监听器，最后调用函数 initUser()初始化用户信息。

```
01  @Override
02  public void onCreate(Bundle savedInstanceState)
03  {
04      super.onCreate(savedInstanceState);
05      setContentView(R.layout.login);
06      mContext=this;
07      //实例化数据库操作类
08      dbHelper = new DataHelper(this);
09      icon=(ImageView)findViewById(R.id.icon);
10      ImageButton select=(ImageButton)findViewById(R.id.iconSelectBtn);
11      Button login=(Button)findViewById(R.id.login);
12      username=(EditText)findViewById(R.id.iconSelect);
13      //绑定监听器
14      select.setOnClickListener(selectClickListener);
15      login.setOnClickListener(loginClickListener);
16      //初始化底部菜单
17      initGridViewMenu();
18      //初始化菜单监听器
19      initMenuListener();
20      //初始化用户信息
21      initUser();
22  }
```

16.6.2　用户的授权

如下所示为函数 initUser()的实现代码，程序先是通过 dbHelper.GetUserList(false)获得所有授权用户的信息，注意这里传入的参数是 false，也就是获得信息中包含昵称和图标的信息。接着判断 userList 是否为空，为空表明当前没有授权的用户，将弹出对话框提示用户是否进行授权，单击“确定”按钮将进入授权界面进行授权。

如果 userList 不为空，表明当前已有授权用户信息，获取存储在 select_name 文件中的 name 字段的值，如果不为空，则通过 GetUserByName()函数取得用户的信息，并调用 setIconAndText 将当前用户的昵称和图标显示到指定位置。如果获取不到当前用户的信息，就用一个默认值代替。

```
01  //初始化用户信息
02  private void initUser(){
03      //获取账号列表
04      userList = dbHelper.GetUserList(false);
05      if(userList.isEmpty())
06      {
07          Builder mBuilder=new AlertDialog.Builder(mContext);
08          mBuilder.setTitle("提示");
09          mBuilder.setMessage("您还未创建任何账户，是否现在创建？");
10          mBuilder.setPositiveButton("确定", new android.content.
            DialogInterface.OnClickListener(){
```

```
11              @Override
12              public void onClick(DialogInterface dialog, int which) {
13                  // TODO Auto-generated method stub
14                  Intent intent = new Intent();
15                  intent.setClass(LoginActivity.this, AuthorizeActivity.
                    class);
16                  startActivity(intent);
17                  LoginActivity.this.finish();
18              }
19          }).setNegativeButton("取消", new android.content.
            DialogInterface.OnClickListener(){
20              @Override
21              public void onClick(DialogInterface dialog, int which) {
22                  // TODO Auto-generated method stub
23                  LoginActivity.this.finish();
24              }
25          });
26          mBuilder.create().show();
27      }
28      else
29      {
30          //记录当前选择的用户名称
31          SharedPreferences preferences = getSharedPreferences
            ("select_name", Activity.MODE_PRIVATE);
32          String str= preferences.getString("name", "");
33          if(str!="")
34          {
35              //从数据库中选择对应的数据
36              selectedUser=dbHelper.GetUserByName(str);
37          }
38          //设置当前显示的用户昵称和图标
39          setIconAndText();
40      }
41  }
42  //设置当前显示的用户昵称和图标
43  public void setIconAndText()
44  {
45      if(selectedUser == null)
46      {
47          //如果当前选择的是空，则取数据库的第一条数据
48          selectedUser=userList.get(0);
49      }
50      if(selectedUser.getUserName() == null)
51      {
52          //若数据库中没有相应的用户信息，则设置为“未知”
53          username.setText("未知");
54      }else{
55          username.setText(selectedUser.getUserName());
56      }
57      if(selectedUser.getUserIcon() != null)
58      {
59          icon.setImageDrawable(selectedUser.getUserIcon());
60      }else{
61          //若数据库中没有相应用户的图标，则设置用户图标为默认图标
62          icon.setImageDrawable(mContext.getResources().getDrawable
            (R.drawable.default_icon));
63      }
64  }
```

16.6.3　按键监听器的设置

初始化完界面之后，要处理的就是那些按键监听器了。先看一下绿色图标按钮对应的监听器 selectClickListener 的实现，当单击按键时先通过 LayoutInflater 膨胀出用户列表视图，设置适配器 adapter，并为列表设置按键监听器 listener。

```
01  OnClickListener selectClickListener =new OnClickListener(){
02      @Override
03      public void onClick(View v) {
04          // TODO Auto-generated method stub
05              AlertDialog.Builder builder;
06              //膨胀出用户列表视图
07              LayoutInflater inflater = (LayoutInflater) mContext.
                getSystemService(LAYOUT_INFLATER_SERVICE);
08              View layout = inflater.inflate(R.layout.user_list,null);
09              ListView myListView = (ListView) layout.
                findViewById(R.id.userList);
10              //为列表元素绑定按键监听器
11              myListView.setOnItemClickListener(listener);
12              Log.e("LoginActivity","userlist-size:"+userList.size());
13              //设置数据适配器
14              UserAdapter adapter = new UserAdapter(mContext, userList);
15              myListView.setAdapter(adapter);
16              builder = new AlertDialog.Builder(mContext);
17              builder.setView(layout);
18              alertDialog = builder.create();
19              //显示对话框
20              alertDialog.show();
21      }
22  };
```

16.6.4　用户适配器

这里用到了用户数据适配器 UserAdapter，UserAdapter 的实现代码如下所示，继承于 BaseAdapter，并实现 BaseAdapter 提供的对应接口即可。

```
01  package com.guo.weibo;          //声明包语句
02~11 行为引入相关类，这里不再列举，请阅读光盘内容
//……
12  //用户信息适配器
13  public class UserAdapter extends BaseAdapter{
14      private List<UserInfo> userList;
15      private Context mContext;
16      //构造函数
17       public UserAdapter(Context context,List<UserInfo> info)
18       {
19          mContext=context;
20          userList=info;
21       }
22       //获得列表的元素个数
23      @Override
24      public int getCount() {
25          return userList.size();
26      }
```

```
27      //获得指定位置的元素
28      @Override
29      public Object getItem(int position) {
30          return userList.get(position);
31      }
32      //获得位置信息
33      @Override
34      public long getItemId(int position) {
35          return position;
36      }
37       //获得每个元素的视图
38      @Override
39      public View getView(int position, View convertView, ViewGroup parent){
40          convertView = LayoutInflater.from(mContext.
            getApplicationContext()).inflate(R.layout.item_user, null);
41          ImageView iv = (ImageView) convertView.
            findViewById(R.id.iconImg);
42          TextView tv = (TextView) convertView.findViewById(R.id.name);
43          UserInfo user = userList.get(position);
44          try {
45          //设置图片显示
46          iv.setImageDrawable(user.getUserIcon());
47          //设置信息
48          tv.setText(user.getUserName());
49          } catch (Exception e) {
50          e.printStackTrace();
51          }
52          return convertView;
53      }
```

16.6.5　对话框监听器

设置弹出对话框按键监听器的代码如下所示，当用户单击任一元素时，将对应行的用户信息保存到 selectedUser 中，并设置当前显示的用户昵称和图标并关闭对话框。

```
01  OnItemClickListener listener=new OnItemClickListener(){
02      @Override
03      public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
04              long arg3) {
05          // TODO Auto-generated method stub
06          //将当前选中的用户信息保存到 selectedUser
07          selectedUser = userList.get(arg2);
08          //设置当前显示的用户昵称和图标
09          setIconAndText();
10          alertDialog.dismiss();
11      }
12  };
```

16.6.6　底部菜单的实现

接着我们来看一下底部菜单的显示和功能的实现，代码如下所示，initGridView 实现了底部菜单初始化的所有工作，包括设置背景图片、设置选中后的背景图片、设置列数、设置适配器等。通过函数 getMenuAdapter()为底部菜单新建一个适配器，将图片和标题分

别对应地显示到底部菜单中。

代码 38～62 行实现了对菜单项监听器的设置，单击“添加”直接跳转到认证的界面，单击“退出”则直接关闭本界面，单击“删除”则调用 delUser()函数，先是删除当前的用户在数据库中的信息，然后重新执行界面的初始化函数 initUser()。

```
01    /**为 GridView 配置菜单资源*/
02    private void initGridViewMenu(){
03       mGridView = (GridView)findViewById(R.id.gridview_toolbar);
04       //设置选中时候的背景图片
05       mGridView.setSelector(R.drawable.menu item selected);
06       //设置背景图片
07       mGridView.setBackgroundResource(R.drawable.menu_background);
08       //设置选中后的背景图片
09       mGridView.setSelector(R.drawable.menu item selected);
10       //设置列数
11       mGridView.setNumColumns(3);
12       //设置居中对齐
13       mGridView.setGravity(Gravity.CENTER);
14       //设置水平、垂直间距为 10
15       mGridView.setVerticalSpacing(10);
16       mGridView.setHorizontalSpacing(10);
17       //设置适配器
18       mGridView.setAdapter(getMenuAdapter(gridview_menu_title,
         girdview_menu_image));
19     }
20    /**菜单适配器*/
21    private SimpleAdapter getMenuAdapter(String[] menuNameArray,
22          int[] imageResourceArray) {
23      //数组列表用于存放映射表
24      ArrayList<HashMap<String, Object>> mData = new ArrayList<HashMap
        <String, Object>>();
25      for (int i = 0; i < menuNameArray.length; i++) {
26         HashMap<String, Object> mMap = new HashMap<String, Object>();
27         //将 image 映射成图片资源
28         mMap.put("image", imageResourceArray[i]);
29         //将 title 映射成标题
30         mMap.put("title", menuNameArray[i]);
31         mData.add(mMap);
32      }
33       //新建简单适配器，设置适配器的布局文件、映射关系
34       SimpleAdapter mAdapter = new SimpleAdapter(this, mData,R.layout.
         login_item_menu, new String[] { "image", "title" },
         new int[] { R.id.item_image, R.id.item_text });
35      return mAdapter;
36  }
37    /**菜单项的监听*/
38    protected void initMenuListener(){
39      mGridView.setOnItemClickListener(new OnItemClickListener(){
40
41         public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
42                long arg3) {
43            switch(arg2){
44            //添加
45            case 0:
46               Intent intent = new Intent();
47               intent.setClass(LoginActivity.this, AuthorizeActivity.
                 class);
```

```
48                  startActivity(intent);
49                  LoginActivity.this.finish();
50                  break;
51              //退出
52              case 1:
53                  LoginActivity.this.finish();
54                  break;
55              //删除
56              case 2:
57                  delUser();
58                  break;
59              }
60          }
61      });
62    }
63    //删除当前用户
64    public void delUser()
65    {
66      if(selectedUser.getUserId() != null)
67      {
68          dbHelper.DelUserInfo(selectedUser.getUserId());
69          //更新当前界面
70          initUser();
71      }
72    }
```

16.6.7　“登录”按钮

最后来到最关键的“登录”按钮，这里有一句话很关键，那就是 Utility.setAuthorization(new OAuth2AccessTokenHeader())，刚开始笔者正是没有加这一句话导致一直无法自动登录，找了很多原因，最后才在网上找到解决办法，而这一句只有 SDK 的例子中没有，只有比较熟悉 Oauth 2.0 的认证过程才能够定位到这个问题。

设置完认证方式，再设置 access_token 就完成了用户的自动登录。接着我们利用刚才获得的 access_token 就可以调用 API 进行一些相关的操作了，如代码 14 行所示。我们登录完成后的第一件事是更新用户的昵称和图标，同时这也可以检查 access_token 的有效性。

因为 access_token 对于未审核的应用只有一天的有效期，因此经常会出现过期的现象，我们通过返回的 JSON 数据中是否包含 error 并且 error_code 为 21315 就可以判断 access_token 是否已经过期，其他的错误代码也分别会代表一种错误类型。读者可以在 API 使用的页面找到相关的介绍。

如果更新过程没有出错，说明我们这个 access_token 是有效的，调用 GoHome()进入用户主页。

```
01  OnClickListener loginClickListener =new OnClickListener(){
02      @Override
03      public void onClick(View v) {
04          // TODO Auto-generated method stub
05          //设置认证方式
06          Utility.setAuthorization(new Oauth2AccessTokenHeader());
07          //设置 access token
08          String token=selectedUser.getToken();
09          AccessToken accessToken = new AccessToken(token, ConstParam.
            CONSUMER_SECRET);
```

```
10          Common.weibo.setAccessToken(accessToken);
11          weibo=Common.weibo;
12          user id=selectedUser.getUserId();
13          //更新当前用户的昵称和图标
14          if(updateNicknameAndIcon())
15          {
16              //跳转到用户主界面
17              GoHome();
18              LoginActivity.this.finish();
19          }
20      }
21  };
22   //进入用户首页
23    private void GoHome(){
24      if(userList!=null)
25      {
26          if(selectedUser !=null)
27          {
28              //获取当前选择的用户并且保存
29              Common.login user=selectedUser;
30              //进入用户首页
31              Intent intent = new Intent();
32              intent.setClass(LoginActivity.this, HomeActivity.class);
33              startActivity(intent);
34          }
35      }
36    }
37    //获得昵称和用户图标
38    public boolean updateNicknameAndIcon()
39    {
40      try {
41          //获得用户的详细信息
42          JSONObject json=getUserDetail(mContext,weibo,user_id);
43          Log.e("guojs-json","ddd"+json.toString());
44          //如果返回的数据含有 error，并且错误代码为“21315”，说明授权已过期
45          if(json.has("error") && json.getString("error code")
             == "21315")
46          {
47             Toast.makeText(mContext, "授权过期，请重新授权！",
               Toast.LENGTH_LONG).show();
48             return false;
49              //其他错误类型
50          }else if(json.has("error")){
51             Toast.makeText(mContext, "出错了，错误代码为:
               "+json.getString("error_code"), Toast.LENGTH_LONG).show();
52             return false;
53             //如果没有包含错误信息，说明信息正确
54          }else{
55              //获得用户昵称
56              String nickName = json.getString("screen_name");
57              Log.e("guojs","nickname:"+nickName);
58              //获得用户图标地址，并下载图标
59              Bitmap bm=Common.getBm(new URL(json.getString
                ("profile image url")));
60              //更新数据库的昵称和图标信息
61              dbHelper.UpdateUserInfo(nickName,bm,user id);
62              dbHelper.Close();
63              return true;
64          }
```

```
65      } catch (Exception e) {
66          Log.e("guojs",e.getMessage());
67          return false;
68      }
69  }
70   //调用 API，获得用户信息
71   public JSONObject getUserDetail(Context mContext,Weibo weibo,
     Stringuid)
72   {
73       String url=Weibo.SERVER+"users/show.json";
74       WeiboParameters bundle=new WeiboParameters();
75       //添加 source 和 uid 参数
76       bundle.add("source", ConstParam.CONSUMER KEY);
77       bundle.add("uid",uid);
78       String rlt="";
79         try {
80             rlt = weibo.request(mContext, url, bundle, "GET", weibo.
               getAccessToken());
81              return new JSONObject(rlt);
82         } catch (Exception e) {
83              // TODO Auto-generated catch block
84              e.printStackTrace();
85              return null;
86         }
87   }
```

16.7　用户首页设计

终于来到显示用户主页面了，在这里用户可以看到关注的人的最新微博，如图 16.12 所示。



图 16.12　用户首页

（1）在 res/layout 中新建 home.xml，代码如下所示。从上到下主要由两部分组成，最上面是标题部分，用于显示用户的昵称和两个 ImageButton。右边两个按钮功能分别而是写微博和刷新页面，中间部分是整个页面的主体，由一个 ListView 和一个用来显示进度条的 RelativeLayout 组成。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout
03      xmlns:android="http://schemas.android.com/apk/res/android"
04      android:id="@+id/layout"
05      android:background="@drawable/background"
06      android:orientation="vertical"
07      android:layout_width="fill_parent"
08      android:layout_height="fill_parent">
09      <!-- 最上面一行 -->
10      <RelativeLayout
11          android:layout width="fill parent"
12          android:layout height="wrap content"
13          android:layout margin="3px">
14          <!-- 用于显示用户昵称 -->
15          <TextView
16              android:id="@+id/showName"
17              android:layout width="wrap content"
18              android:layout height="wrap content"
19              android:layout centerInParent="true"
20              android:textColor="#343434"
21              android:textSize="15px" />
22              <!-- 用于显示写微博图标 -->
23          <ImageButton
24              android:id="@+id/writeBtn"
25              android:layout_width="50px"
26              android:layout_height="50px"
27              android:layout_toLeftOf="@+id/refreshBtn"
28              android:background="@drawable/write" />
29              <!-- 用于显示刷新图标 -->
30          <ImageButton
31              android:id="@+id/refreshBtn"
32              android:layout width="50px"
33              android:layout height="50px"
34              android:layout alignParentRight="true"
35              android:layout marginLeft="12px"
36              android:background="@drawable/refresh" />
37      </RelativeLayout>
38          <!-- 黑色分隔线 -->
39        <LinearLayout
40            android:layout width="fill parent"
41            android:layout height="1px"
42            android:background="#000000" />
43            <!-- 微博显示主体 -->
44      <RelativeLayout
45      android:layout_width="fill_parent"
46      android:layout_height="fill_parent">
47          <!-- 微博列表 -->
48          <ListView
49              android:id="@+id/Msglist"
50              android:layout width="fill parent"
51              android:layout height="match parent"
52              android:dividerHeight="2px"
53              android:layout_margin="0px"
```

```
54              android:background="#BBFFFFFF"
55              android:cacheColorHint="#00000000"
56              android:layout above="@+id/toolbarLayout"
57              android:fastScrollEnabled="true"
58              android:focusable="true" />
59          <!-- 显示载入动画 -->
60          <LinearLayout
61              android:id="@+id/loadingLayout"
62              android:layout_width="wrap_content"
63              android:layout_height="wrap_content"
64              android:orientation="vertical"
65              android:visibility="visible"
66              android:layout centerInParent="true">
67              <!--载入进度条 -->
68              <ProgressBar
69                  android:id="@+id/loading"
70                  android:layout width="47px"
71                  android:layout height="47px"
72                  android:layout gravity="center"
73                  style="@style/progressStyle" />
74              <TextView
75                  android:layout_width="wrap_content"
76                  android:layout_height="wrap_content"
77                  android:text="正在载入"
78                  android:textSize="12px"
79                  android:textColor="#9c9c9c"
80                  android:layout gravity="center"
81                  android:layout below="@+id/loading" />
82          </LinearLayout>
83      </RelativeLayout>
```

（2）用于显示微博内容的界面如下所示，整个布局采用水平排列的 LinearLayout，左边是一个 ImageView 用于显示用户的图标。右边分为两部分，上面从左到右分别显示用户的昵称、微博内容是否包含图片的标志、微博的发布时间，下面则显示微博的内容。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout
03  xmlns:android="http://schemas.android.com/apk/res/android"
04  android:layout width="wrap content"
05  android:layout height="wrap content"
06  android:orientation="horizontal">
07  <!-- 用户 Icon -->
08  <ImageView
09      android:id="@+id/wbicon"
10      android:layout_width="50px"
11      android:layout_height="50px"
12      android:layout_margin="8px" />
13      <!-- 微博内容 -->
14      <LinearLayout
15          android:layout width="fill parent"
16          android:layout height="wrap content"
17          android:orientation="vertical"
18          android:paddingLeft="0px"
19          android:paddingRight="5px"
20          android:layout marginTop="5px"
21          android:layout marginBottom="5px">
22          <!-- 微博标题部分 -->
23          <RelativeLayout
24              android:layout_width="fill_parent"
```

```
25          android:layout_height="wrap_content">
26          <!-- 用户昵称 -->
27          <TextView
28             android:id="@+id/wbuser"
29             android:layout_width="wrap_content"
30             android:layout_height="wrap_content"
31             android:textSize="15px"
32             android:textColor="#424952"
33             android:layout_alignParentLeft="true" />
34             <!-- 微博是否包含图片的标志，若有图片则显示，否则不显示 -->
35          <ImageView
36             android:id="@+id/wbimage"
37             android:layout_width="wrap_content"
38             android:layout_height="wrap_content"
39             android:layout_marginTop="3px"
40             android:layout_marginRight="5px"
41             android:layout_toLeftOf="@+id/wbtime" />
42             <!--微博发布的时间-->
43          <TextView
44             android:id="@+id/wbtime"
45             android:layout_width="wrap_content"
46             android:layout_height="wrap_content"
47             android:layout_alignParentRight="true"
48             android:textColor="#f7a200"
49             android:textSize="12px" />
50       </RelativeLayout>
51       <!-- 微博内容 -->
52       <TextView
53          android:id="@+id/wbtext"
54          android:layout_width="wrap_content"
55          android:layout_height="wrap_content"
56          android:textColor="#424952"
57          android:textSize="13px"
58          android:layout_marginTop="4px" />
59    </LinearLayout>
60 </LinearLayout>
```

（3）在微博载入过程中需要显示一个载入动画，我们在 res/anim 下新建 loading.xml，代码如下所示：

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <animation-list android:oneshot="false"
03 xmlns:android="http://schemas.android.com/apk/res/android">
04 <item android:duration="200" android:drawable="@drawable/r1" />
05 <item android:duration="200" android:drawable="@drawable/r2" />
06 <item android:duration="200" android:drawable="@drawable/r3" />
07 <item android:duration="200" android:drawable="@drawable/r4" />
08 <item android:duration="200" android:drawable="@drawable/r5" />
09 <item android:duration="200" android:drawable="@drawable/r6" />
10 <item android:duration="200" android:drawable="@drawable/r7" />
11 <item android:duration="200" android:drawable="@drawable/r8" />
```

（4）接着在 res/values 下新建 loadingstyles.xml，代码如下所示：

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <resources>
03 <style
04    name="progressStyle"
05    width="wrap_content"
06    height="wrap_content"
```

```
07      parent="@android:style/Widget.ProgressBar.Large"
08      mce bogus="1">
09      <item name="android:indeterminateDrawable">@anim/loading</item>
10   </style>
```

16.8　用户首页功能实现

（1）新建 HomeActivity.java，代码如下所示，在 onCreate()函数中先初始化界面元素，并且顶部的两个 ImageView 绑定监听器，之后执行 loadList()函数初始化界面。这里注意的一点是，最上面的两个是 ImageButton 而不是 Button，在初始化的时候需要注意，虽然它们的用法很相似，但不能混为一谈。

```
01  public class HomeActivity extends Activity{
02      //保存当前上下文
03      Context mContext;
04      //存储所有微博信息
05      private List<WeiboInfo> wbList;
06      //载入动画
07      LinearLayout loadingLayout;
08      Weibo weibo;
09      @Override
10      public void onCreate(Bundle savedInstanceState)
11     {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.home);
14         loadingLayout=(LinearLayout)findViewById(R.id.loadingLayout);
15         //写微博
16         ImageButton btn_write=(ImageButton)findViewById(R.id.writeBtn);
17         btn_write.setOnClickListener(new OnClickListener(){
18             @Override
19             public void onClick(View v) {
20                 // TODO Auto-generated method stub
21                 try {
22                     //写微博
23                     weibo.share2weibo(HomeActivity.this, weibo.
                       getAccessToken().getToken(), weibo.getAccessToken()
24                             .getSecret(), "abc", "");
25                 } catch (WeiboException e) {
26                     // TODO Auto-generated catch block
27                     e.printStackTrace();
28                 }
29             }
30         });
31         //刷新界面
32         ImageButton refresh=(ImageButton)findViewById(R.id.refreshBtn);
33         refresh.setOnClickListener(new OnClickListener(){
34             @Override
35             public void onClick(View v) {
36                 // TODO Auto-generated method stub
37                 loadList();
38             }
39         });
40         mContext=this;
41         //初始化界面
42         loadList();
43
```

（2）接下来我们分析一下 HomeActivity 的核心函数 loadList()，代码如下所示，一开始先判断用户是否已经登录，如果已登录，则将登录用户的信息保存到 user 变量中，并将当前用户的昵称显示到最上方。

由于登录过程已经完成了对 access_token 的设置，因此直接获得 Common.weibo 并传入函数 getFriendsTimeline 即可向服务器取得相应信息。同样，因为获得的过程是一个多个环节组成的过程，会有很多因素如网络连接不稳定、参数错误等导致得不到想要的数据，因此对返回的结果进行错误判断是很有必要的。这里并没有将错误的类型逐一枚举出来进行处理，只将典型的“授权过期”这种错误类型列举出来，其他的类型读者可以根据需要自行处理。

通过分析新浪微博的 API 数据可以发现，微博的相关信息都存储在 statues 里面，因为我们先获得 statues，对该 JSON 对象进行处理。statues 是一个 JSON 数组，通过遍历这个数组的内容，可以取出每一条微博信息，分别转换成 JSON 对象。

接着从得到的 JSON 对象中获得所需要的值，如 screen_name、profile_image_url、text 等保存到 WeiboInfo 类中，并将一个个 WeiboInfo 类添加到 wbList 中。然后新建 WeiboAdapter 将 WeiboInfo 的数据适配到对应的视图中，并为每一行元素设置按键监听器，当单击每一行元素的时候可以查看微博的详细信息。此外，因为此时已经载入完成，因此需要将载入进度条设置成 GONE。

```
001 //初始化界面显示
002 private void loadList(){
003     //如果当前没有用户登录直接返回
004        if(Common.login_user==null)
005        {
006             return;
007        }
008        else
009        {
010         //取得当前登录的用户
011            UserInfo user=Common.login user;
012            //显示当前用户名称
013            TextView showName=(TextView)findViewById(R.id.showName);
014            showName.setText(user.getUserName());
015         weibo=Common.weibo;
016         //取得当前用户关注的用户的最新微博信息
017         JSONObject json = getFriendsTimeline(mContext,weibo);
018         Log.e("guojs",json.toString());
019         JSONArray data;
020         try {
021            //判断 access_token 是否过期
022            if(json.has("error") && json.getString("error_code") == "21315")
023            {
024            Toast.makeText(mContext, "授权过期，请重新授权！", Toast.
               LENGTH LONG).show();
025            //如果过期则引导用户重新授权
026             Intent it=new Intent(HomeActivity. this,LoginActivity.
027             startActivity(it);
028                HomeActivity.this.finish();
029                //判断返回值是否出错
030             }else if(json.has("error")){
031               Toast.makeText(mContext, "出错了，错误代码为："+json.
                  getString("error_code"), Toast.LENGTH_LONG).show();
```

```
032                 //出错则引导用户重新登录
033                 Intent it=new Intent(HomeActivity. this,LoginActivity.
                    class);
034                 startActivity(it);
035                 HomeActivity.this.finish();
036             }else{
037                 //取得 statuses 信息
038                 data = new JSONArray(json.getString("statuses"));
039                 Log.e("home-length","changdu :"+data.length());
040                 //遍历每个信息
041                 for(int i=0;i<data.length();i++)
042                 {
043                     JSONObject d=data.getJSONObject(i);
044                     if(d!=null){
045                         //取得每条信息中包含的用户的信息
046                         JSONObject u=d.getJSONObject("user");
047                         //微博 id
048                         String id=d.getString("id");
049                         //取得用户 id
050                         String userId=u.getString("id");
051                         //取得用户昵称
052                         String userName=u.getString("screen name");
053                         //取得用户图标地址
054                         String userIcon=u.getString
                        ("profile_image_url");
055                         Log.e("userIcon", userIcon);
056                         //取得微博发布时间
057                         String time=d.getString("created at");
058                         //取得微博内容
059                         String text=d.getString("text");
060                         Boolean haveImg=false;
061                         //判断当前微博内容是否包含图片
062                         if(d.has("thumbnail pic")){
063                             //如果有图片，则设置图片标志位
064                             haveImg=true;
065                         }
066                         Date date=new Date(time);
067                         //转换成几分钟前、几小时前……
068                         time=getTimeDiff(date);
069                         if(wbList == null){
070                             //如果还没有初始化微博列表，则进行初始化
071                             wbList = new ArrayList<WeiboInfo>();
072                         }
073                         //实例化一个 WeiboInfo 类，用于保存微博的信息
074                         WeiboInfo w=new WeiboInfo();
075                         w.setId(id);
076                         w.setUserId(userId);
077                         w.setUserName(userName);
078                         w.setTime(time);
079                         w.setText(text);
080
081                         w.setHaveImage(haveImg);
082                         w.setUserIcon(userIcon);
083                         //将信息添加到微博数组中
084                         wbList.add(w);
085                     }
086                 }
087             }
088         } catch (JSONException e) {
```

```
089             // TODO Auto-generated catch block
090             e.printStackTrace();
091         }
092         //判断微博列表是否为空
093            if(wbList!=null)
094            {
095             //创建微博信息列表的适配器
096                WeiboAdapater adapater = new WeiboAdapater();
097                ListView Msglist=(ListView)findViewById(R.id.Msglist);
098                //设置列表的按键监听器
099                Msglist.setOnItemClickListener(new OnItemClickListener (){
100                    @Override
101                    public void onItemClick(AdapterView<?> arg0,
                       View view,int arg2, long arg3) {
102                     //获得视图绑定的 tag 信息
103                        Object obj=view.getTag();
104                        if(obj!=null){
105                            String id=obj.toString();
106                            //启动查看微博详细信息的视图
107                            Intent intent = new Intent(HomeActivity.
                            this,ViewActivity.class);
108                            //将微博的 id 绑定到 b 中发送给下一个 Activity
109                            Bundle b=new Bundle();
110                            b.putString("key", id);
111                            intent.putExtras(b);
112                            startActivity(intent);
113                        }
114                    }
115
116                });
117                Msglist.setAdapter(adapater);
118            }
119        }
120        //隐藏载入进度条
121        loadingLayout.setVisibility(View.GONE);
122    }
123 //获得用户关注的用户的最新微博信息
124 public JSONObject getFriendsTimeline(Context mContext,Weibo weibo)
125 {
126      String url=Weibo.SERVER+"statuses/friends timeline.json";
127      WeiboParameters bundle=new WeiboParameters();
128      //传入 source
129      bundle.add("source", ConstParam.CONSUMER KEY);
130      String rlt="";
131        try {
132            rlt = weibo.request(mContext, url, bundle, "GET", weibo.)
               getAccessToken());
133            return new JSONObject(rlt);
134        } catch (Exception e) {
135            // TODO Auto-generated catch block
136            e.printStackTrace();
137            return null;
138        }
139 }
140 //取得传入时间与当前时间的时间差
141 public String getTimeDiff(Date date) {
142         Calendar cal = Calendar.getInstance();
143         long diff = 0;
144         Date dnow = cal.getTime();
145         String str = "";
```

```
146         diff = dnow.getTime() - date.getTime();
147         //30 * 24 * 60 * 60 * 1000=2592000000 毫秒
148         if  (diff > 2592000000L) {
149             str="1 个月前";
150           //21 * 24 * 60 * 60 * 1000=1814400000 毫秒
151         } else if  (diff > 1814400000) {
152             str="3 周前";
153           //14 * 24 * 60 * 60 * 1000=1209600000 毫秒
154         } else if  (diff > 1209600000) {
155             str="2 周前";
156           //7 * 24 * 60 * 60 * 1000=604800000 毫秒
157         } else if  (diff > 604800000) {
158             str="1 周前";
159             //24 * 60 * 60 * 1000=86400000 毫秒
160         } else if (diff > 86400000) {
161             // 60 * 60 * 1000=3600000 毫秒
162             str=(int)Math.floor(diff/86400000f) + "天前";
163         } else if (diff > 3600000 ) {
164             //1 * 60 * 1000=60000 毫秒
165             str=(int)Math.floor(diff/3600000f) + "小时前";
166         } else if (diff > 60000) {
167             str=(int)Math.floor(diff/60000) +"分钟前";
168         }else{
169             str=(int)Math.floor(diff/1000) +"秒前";
170         }
171         return str;
172     }
```

（3）WeiboInfo 类的实现代码如下所示，这个类比较简单，因为只用来存取，只要几个 get、set 函数和对应的成员属性即可。

```
01  package com.guo.weibo;
02  public class WeiboInfo {
03      //微博 id
04      private String id;
05      //发布人 id
06      private String userId;
07      //发布人名字
08      private String userName;
09      //发布人头像
10      private String userIcon;
11      //发布时间
12      private String time;
13      //是否有图片
14      private Boolean haveImage=false;
15      //文章内容
16      private String text;
17      //取得微博 id
18      public String getId(){
19          return id;
20      }
21      //设置微博 id
22      public void setId(String id){
23          this.id=id;
24      }
25      //取得用户 id
```

```
26      public String getUserId(){
27          return userId;
28      }
29      //设置用户 id
30      public void setUserId(String userId){
31          this.userId=userId;
32      }
33      //取得用户昵称
34      public String getUserName(){
35          return userName;
36      }
37      //设置用户昵称
38      public void setUserName(String userName){
39          this.userName=userName;
40      }
41      //获取用户图标
42      public String getUserIcon(){
43          return userIcon;
44      }
45      //设置用户图标
46      public void setUserIcon(String userIcon){
47          this.userIcon=userIcon;
48      }
49      //获取时间
50      public String getTime(){
51          return time;
52      }
53      //设置时间
54      public void setTime(String time)
55      {
56          this.time=time;
57      }
58      //取得是否包含图片的标志
59      public Boolean getHaveImage(){
60          return haveImage;
61      }
62      //设置是否包含图片的标志
63      public void setHaveImage(Boolean haveImage){
64          this.haveImage=haveImage;
65      }
66      //取得微博内容
67      public String getText(){
68          return text;
69      }
70      //设置微博内容
71      public void setText(String text){
72          this.text=text;
73      }
```

（4）如下所示，为 WeiboAdapter 的实现过程，在 getView()函数中用到了 AsyncImageLoader 类异步加载图片。由于图片是从网络上下载的，比较耗时，而且图片资源一般相对较大，因此这里需要作一些优化处理。

Dalvik VM 给每个进程都分配了一定量的可用堆内存，当我们处理一些耗费资源的操作时可能会产生 OOM 错误（OutOfMemoryError）这样的异常。在 Java 中内存管理，引用分为 4 大类：强引用 HardReference、弱引用 WeakReference、软引用 SoftReference 和虚引用 PhantomReference。它们的区别也很明显，HardReference 对象是即使虚拟机内存吃紧抛

出 OOM 也不会导致这一引用的对象被回收，而 WeakReference 等更适合于一些数量不多，但体积稍微庞大的对象，在这 4 个引用中，它是最容易被垃圾回收的，而我们对于显示类似 Android Market 中每个应用的 App Icon 时可以考虑使用 SoftReference 来解决内存不至于快速回收。同时当内存短缺面临 Java VM 崩溃抛出 OOM 前时，软引用将会强制回收内存。最后的虚引用一般没有实际意义，仅仅观察 GC 的活动状态，对于测试比较实用，同时必须和 ReferenceQueue 一起使用。

对于图片数据，我们通过 HashMap 的方式来添加一组 SoftReference 对象来临时保留数据。当调用 loadDrawable 函数显示图片时，先从 imageCache 中查询是否已有图片的缓存，没有则开启一个新线程下载图片，并发送消息调用接口函数 imageLoaded 将图片显示到界面中。这里我们新建了一个接口 ImageCallback，用于实现图片下载完成之后的更新界面操作。

同时，可以看到我们对显示的文字也作了一些处理，利用 textHighlight()函数高亮显示关键字，主要用到了正则匹配。

```
001 //微博列表 Adapater
002 public class WeiboAdapater extends BaseAdapter{
003      //获取列表总行数
004     @Override
005     public int getCount() {
006         return wbList.size();
007     }
008     //取得当前选中的微博
009     @Override
010     public Object getItem(int position) {
011         return wbList.get(position);
012     }
013     //取得当前的位置
014     @Override
015     public long getItemId(int position) {
016         return position;
017     }
018      //得到每一行的视图
019     @Override
020     public View getView(int position, View convertView, ViewGroup parent){
021         //异步获取图片类
022         AsyncImageLoader asyncImageLoader = new AsyncImageLoader();
023         //膨胀出微博视图
024         convertView = LayoutInflater.from(getApplicationContext()).
            inflate(R.layout.weibo, null);
025         WeiboHolder wh = new WeiboHolder();
026         //取得微博界面的元素
027         wh.wbicon = (ImageView) convertView.findViewById(R.id.wbicon);
028         wh.wbtext = (TextView) convertView.findViewById(R.id.wbtext);
029         wh.wbtime = (TextView) convertView.findViewById(R.id.wbtime);
030         wh.wbuser = (TextView) convertView.findViewById(R.id.wbuser);
031         wh.wbimage=(ImageView) convertView.findViewById(R.id.wbimage);
032         //取得微博信息
033         WeiboInfo wb = wbList.get(position);
034         if(wb!=null){
035             //将微博的 id 保存到标签汇总
036             convertView.setTag(wb.getId());
037             wh.wbuser.setText(wb.getUserName());
038             wh.wbtime.setText(wb.getTime());
```

```
039             wh.wbtext.setText(textHighlight(wb.getText()),
                TextView.BufferType.SPANNABLE);
040             //如果微博内容含有图片，则显示标志图片
041             if(wb.getHaveImage()){
042                 wh.wbimage.setImageResource(R.drawable.images);
043             }
044             //取得用户图标
045             Drawable cachedImage=asyncImageLoader. loadDrawable
                (wb.getUserIcon(),wh.wbicon, new ImageCallback(){
046                 //调用接口显示图片
047                 @Override
048                 public void imageLoaded(Drawable imageDrawable,ImageView
                    imageView, String imageUrl) {
049                     imageView.setImageDrawable(imageDrawable);
050                 }
051             });
052             if (cachedImage == null) {
053                 wh.wbicon.setImageResource(R.drawable.default_icon);
054             }else{
055                 wh.wbicon.setImageDrawable(cachedImage);
056             }
057         }
058         return convertView;
059     }
060     //用于存放微博视图元素的类
061     class WeiboHolder{
062         //微博是否包含图片标志
063         public ImageView wbimage;
064         //用户昵称
065         public TextView wbuser;
066         //微博发布时间
067         public TextView wbtime;
068         //微博内容
069         public TextView wbtext;
070         //用户图标
071         public ImageView wbicon;
072     }
073 }
074 public static class AsyncImageLoader {
075     //SoftReference 是软引用，是为了更好地使系统回收变量
076     private HashMap<String, SoftReference<Drawable>> imageCache;
077     public AsyncImageLoader() {
078         imageCache = new HashMap<String, SoftReference<Drawable>>();
079     }
080
081     public Drawable loadDrawable(final String imageUrl,
        final ImageView imageView, final ImageCallback imageCallback){
082         if (imageCache.containsKey(imageUrl)) {
083             //从缓存中获取
084             SoftReference<Drawable> softReference =
                imageCache.get(imageUrl);
085             Drawable drawable = softReference.get();
086             if (drawable != null) {
087                 return drawable;
088             }
089         }
090         final Handler handler = new Handler() {
091         public void handleMessage(Message message) {
092             //调用接口显示图片
```

```
093             imageCallback.imageLoaded((Drawable) message.obj,
                imageView,imageUrl);
094         }};
095         //建立一个新的线程下载图片
096         new Thread() {
097             @Override
098             public void run() {
099                 Drawable drawable = loadImageFromUrl(imageUrl);
100                 //将图片放入缓存
101                 imageCache.put(imageUrl, new SoftReference
                    <Drawable>(drawable));
102                 Message message = handler.obtainMessage(0, drawable);
103                 //发送消息更新界面
104                 handler.sendMessage(message);
105             }
106         }.start();
107         return null;
108     }
109     //从网上下载图片
110     public static Drawable loadImageFromUrl(String url){
111         URL m;
112         InputStream i = null;
113         try {
114             m = new URL(url);
115             //取得图片的数据转换成 InputStream
116             i = (InputStream) m.getContent();
117         } catch (MalformedURLException e1) {
118             e1.printStackTrace();
119         } catch (IOException e) {
120             e.printStackTrace();
121         }
122         //生成一个 Drawable 对象
123         Drawable d = Drawable.createFromStream(i, "src");
124         return d;
125     }
126 }
127 //回调接口
128 public interface ImageCallback {
129     public void imageLoaded(Drawable imageDrawable,ImageView imageView,
        String imageUrl);
130 }
131 //高亮显示 TextView 的关键文字
132 public SpannableStringBuilder textHighlight(String str)
133 {
134     //高亮显示"#"和"#"之间的内容
135     Pattern pattern =Pattern.compile("#[^#]+#");
136     Matcher matcher = pattern.matcher(str);
137     SpannableStringBuilder style=new SpannableStringBuilder(str);
138     //对所有匹配的字符串进行处理
139     while(matcher.find()){
140         String temp=matcher.group();
141         Log.e("guojs","start"+str.indexOf(matcher.group()));
142         //设置关键字颜色为蓝色
143         style.setSpan(new ForegroundColorSpan(Color.BLUE),
            str.indexOf(temp),str.indexOf(temp)+temp.length(),
            Spannable.SPAN EXCLUSIVE EXCLUSIVE);
144     }
145     //高亮显示"@"和": "之间的文字
146     pattern=Pattern.compile("@[\\w\\W]:");
```

```
147     matcher = pattern.matcher(str);
148     while(matcher.find()){
149        String temp=matcher.group();
150        Log.e("guojs","start"+str.indexOf(matcher.group()));
151        //设置关键字的颜色为蓝色
152        style.setSpan(new ForegroundColorSpan(Color.BLUE),
           str.indexOf(temp),str.indexOf(temp)+temp.length(),
           Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
153     }
154     //高亮显示 http://开头或者 https://开头的文字
155     pattern=Pattern.compile("(http://|https://){1}[\\w\\.\\-/:]+");
156     matcher = pattern.matcher(str);
157     while(matcher.find()){
158        String temp=matcher.group();
159        Log.e("guojs","start"+str.indexOf(matcher.group()));
160        //设置关键字颜色为蓝色
161         style.setSpan(new ForegroundColorSpan(Color.BLUE),
            str.indexOf(temp),str.indexOf(temp)+temp.length(),Spannable.
            SPAN_EXCLUSIVE_EXCLUSIVE);
162     }
163     return style;
164 }
```

（5）关于右上角的两个按钮，“刷新”按钮即重新执行了一遍 loadList()，而单击“写博客”则调用函数 share2weibo，进入发送界面如图 16.13 所示。进入 SDK 查看 share2weibo，发现它调用了 SDK 中的 ShareActivity。而 ShareActivity 中关于发送微博的关键代码如下所示。

如代码 13 行所示，发送代码将调用 upload 这个函数，而 upload 函数的实现方法跟我们之前调用 API 的方法很类似，如代码 50～68 行所示。不同点在于这里用到了一个类 AnsyncWeiboRunner，用于异步执行操作。

```
01 @Override
02   public void onClick(View v) {
03       int viewId = v.getId();
04
05       if (viewId == R.id.btnClose) {
06           finish();
07       } else if (viewId == R.id.btnSend) {
08           Weibo weibo = Weibo.getInstance();
09           try {
10               if (!TextUtils.isEmpty((String)
                 (weibo.getAccessToken().getToken()))) {
11                   this.mContent = mEdit.getText().toString();
12                   if (!TextUtils.isEmpty(mPicPath)) {
13                       upload(weibo, Weibo.getAppKey(), this.mPicPath,
                         this.mContent, "", "");
14
15                   } else {
16                       // Just update a text weibo!
17                       update(weibo, Weibo.getAppKey(), mContent, "", "");
18                   }
19               } else {
20                   Toast.makeText(this, this. getString(R.string.
                     please_login), Toast.LENGTH_LONG);
21               }
22           } catch (MalformedURLException e) {
23               e.printStackTrace();
```

```
24              } catch (IOException e) {
25                  e.printStackTrace();
26              } catch (WeiboException e) {
27                  e.printStackTrace();
28              }
29          } else if (viewId == R.id.ll text limit unit) {
30              Dialog dialog = new AlertDialog.Builder(this).
                setTitle(R.string.attention)
31                      .setMessage(R.string.delete all)
32                      .setPositiveButton(R.string.ok, new DialogInterface.
                        OnClickListener() {
33                          public void onClick(DialogInterface dialog,
                            int which) {
34                              mEdit.setText("");
35                          }
36                      }).setNegativeButton(R.string.cancel, null).create();
37              dialog.show();
38          } else if (viewId == R.id.ivDelPic) {
39              Dialog dialog = new AlertDialog.Builder(this).
                setTitle(R.string.attention)
40                      .setMessage(R.string.del pic)
41                      .setPositiveButton(R.string.ok, new DialogInterface.
                        OnClickListener() {
42                          public void onClick(DialogInterface dialog,
                            int which) {
43                              mPiclayout.setVisibility(View.GONE);
44                          }
45                      }).setNegativeButton(R.string.cancel, null).create();
46              dialog.show();
47          }
48      }
49
50      private String upload(Weibo weibo, String source, String file,
        String status, String lon,
51              String lat) throws WeiboException {
52          WeiboParameters bundle = new WeiboParameters();
53          bundle.add("source", source);
54          bundle.add("pic", file);
55          bundle.add("status", status);
56          if (!TextUtils.isEmpty(lon)) {
57              bundle.add("lon", lon);
58          }
59          if (!TextUtils.isEmpty(lat)) {
60              bundle.add("lat", lat);
61          }
62          String rlt = "";
63          String url = Weibo.SERVER + "statuses/upload.json";
64          AsyncWeiboRunner weiboRunner = new AsyncWeiboRunner(weibo);
65          weiboRunner.request(this, url, bundle, Utility.HTTPMETHOD POST,
            this);
66
67          return rlt;
68      }
69
70      private String update(Weibo weibo, String source, String status, String
        lon, String lat)
71              throws MalformedURLException, IOException, WeiboException {
72          WeiboParameters bundle = new WeiboParameters();
73          bundle.add("source", source);
74          bundle.add("status", status);
75          if (!TextUtils.isEmpty(lon)) {
```

```
76              bundle.add("lon", lon);
77          }
78          if (!TextUtils.isEmpty(lat)) {
79              bundle.add("lat", lat);
80          }
81          String rlt = "";
82          String url = Weibo.SERVER + "statuses/update.json";
83          AsyncWeiboRunner weiboRunner = new AsyncWeiboRunner(weibo);
84          weiboRunner.request(this, url, bundle, Utility.HTTPMETHOD POST,
            this);
85          return rlt;
86      }
```



图 16.13　发送微博界面

16.9　阅读微博界面设计

阅读微博 UI 如图 16.14 所示。

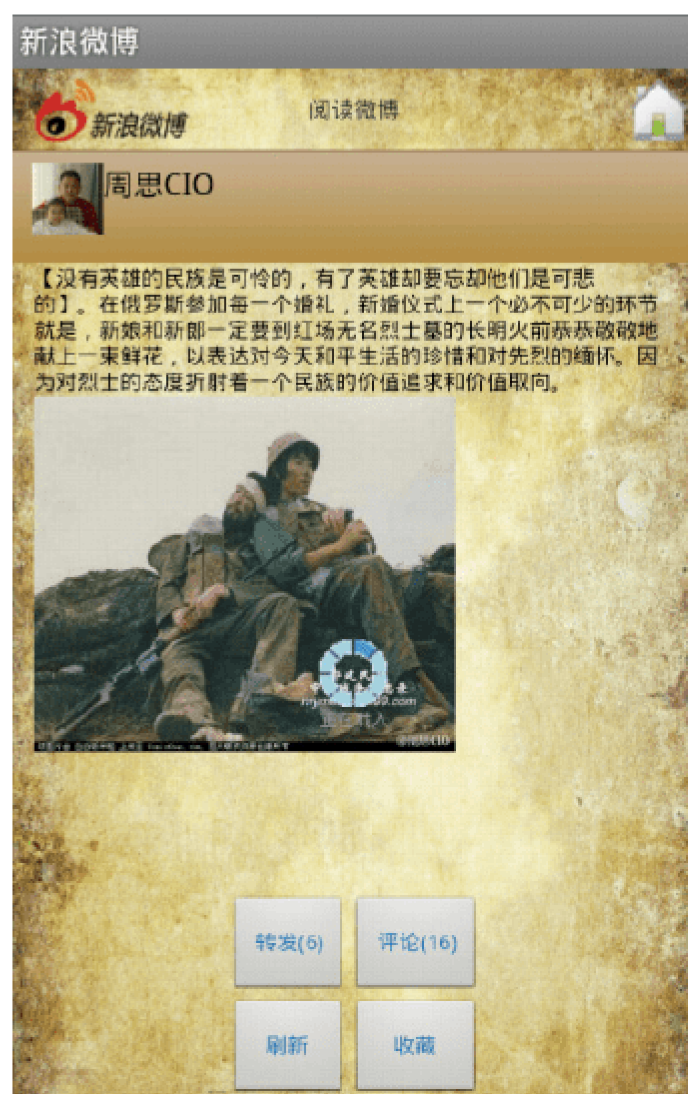


图 16.14　阅读微博界面

在 res/layout 下新建布局文件 weibo_detail.xml，代码如下所示。最上面用 TextView 显示“阅读微博”4 个字，右边是一个 ImageButton，用来返回主页面。下面用一个 LinearLayout 生成一条分隔线，接下去用一个 RelativeLayout 显示用户信息，左边用一个 ImageView 显

示用户头像，右边用一个 TextView 显示用户昵称。

中间是微博的主体，用一个 ScrollView 来实现，上面是微博的文字内容，下面是图片内容。最下面放置了一些按钮，采用 TableLayout 的布局方式。其中中间还有一个 LinearLayout 用来显示载入动画，在载入完成之后将其设置成隐藏。

```
001 <?xml version="1.0" encoding="utf-8"?>
002 <LinearLayout
003   xmlns:android="http://schemas.android.com/apk/res/android"
004   android:id="@+id/layout"
005   android:orientation="vertical"
006   android:background="@drawable/background"
007   android:layout_width="fill_parent"
008   android:layout_height="fill_parent">
009   <!-- 标题显示部分 -->
010   <RelativeLayout
011       android:layout_width="fill_parent"
012       android:layout_height="wrap_content"
013       android:layout_margin="3px" >
014       <!-- 顶部的标题 -->
015       <TextView
016           android:id="@+id/showName"
017           android:layout_width="wrap_content"
018           android:layout_height="wrap_content"
019           android:layout_centerInParent="true"
020           android:textColor="#343434"
021           android:text="阅读微博"
022           android:textSize="16px" />
023       <!-- 返回主页按键 -->
024       <ImageButton
025           android:id="@+id/homeBtn"
026           android:layout_width="50px"
027           android:layout_height="50px"
028           android:layout_alignParentRight="true"
029           android:layout_marginLeft="12px"
030           android:background="@drawable/home" />
031   </RelativeLayout>
032   <!-- 分界线 -->
033   <LinearLayout
034       android:layout_width="fill_parent"
035       android:layout_height="1px"
036       android:background="#6B4226">
037   </LinearLayout>
038   <!-- 用户信息 -->
039   <RelativeLayout
040       android:id="@+id/user_bg"
041       android:layout_width="fill_parent"
042       android:layout_height="78px"
043       android:background="@drawable/content_bg"
044       android:paddingTop="8px"
045       android:paddingLeft="15px" >
046       <!-- 用户头像 -->
047       <ImageView
048           android:id="@+id/usericon"
049           android:layout_width="50px"
050           android:layout_height="50px"
051           android:layout_alignParentLeft="true" />
052           <!-- 用户昵称 -->
053       <TextView
```

```
054          android:id="@+id/username"
055          android:layout width="wrap content"
056          android:layout height="wrap content"
057          android:layout toRightOf="@+id/user icon"
058          android:layout marginLeft="50px"
059          android:textColor="#000000" />
060     </RelativeLayout>
061     <!-- 内容主体部分 -->
062     <RelativeLayout
063       android:layout_width="fill_parent"
064       android:layout_height="fill_parent">
065       <!-- 使用 ScrollView 存放微博内容 -->
066         <ScrollView
067             android:layout width="fill parent"
068             android:layout height="fill parent"
069             android:paddingLeft="17px"
070             android:paddingRight="17px"
071             android:paddingBottom="5px"
072             android:layout above="@+id/menu layout">
073             <LinearLayout
074                 android:layout_width="fill_parent"
075                 android:layout_height="fill_parent"
076                 android:orientation="vertical">
077                 <!-- 微博文本内容 -->
078                 <TextView
079                     android:id="@+id/text"
080                     android:layout width="wrap content"
081                     android:layout height="wrap content"
082                     android:textColor="#000000"
083                     android:textSize="15px" />
084                     <!-- 微博图片 -->
085                 <ImageView
086                     android:id="@+id/pic"
087                     android:layout_width="wrap_content"
088                     android:layout height="wrap content" />
089             </LinearLayout>
090         </ScrollView>
091          <!-- 载入动画 -->
092         <LinearLayout
093             android:id="@+id/loadingLayout"
094             android:layout width="wrap content"
095             android:layout height="wrap content"
096             android:orientation="vertical"
097             android:visibility="visiblity"
098             android:layout_centerInParent="true">
099             <ProgressBar
100                 android:id="@+id/loading"
101                 android:layout_width="47pX"
102                 android:layout_height="47px"
103                 android:layout_gravity="center"
104                 style="@style/progressStyle">
105             </ProgressBar>
106             <TextView
107                 android:layout width="wrap content"
108                 android:layout height="wrap content"
109                 android:text="正在载入"
110                 android:textSize="12px"
111                 android:textColor="#9c9c9c"
112                 android:layout_gravity="center"
113                 android:layout_below="@+id/loading" />
```

```
114         </LinearLayout>
115     <!-- 底部按钮 -->
116         <TableLayout
117             android:id="@+id/menu_layout"
118             android:layout_width="fill_parent"
119             android:layout_height="wrap_content"
120             android:gravity="center"
121             android:layout_alignParentBottom="true"
122             android:layout_marginBottom="5px">
123             <TableRow
124                 android:layout_width="wrap_content"
125                 android:layout_height="wrap_content"
126                 android:gravity="center">
127                 <!–“转发”按钮 -->
128                 <Button
129                     android:id="@+id/btn_gz"
130                     android:layout_width="wrap_content"
131                     android:layout_height="wrap_content"
132                     android:textColor="#3882b8"
133                     android:textSize="15px"
134                     android:text="  转发(1231)"  />
135                 <!–“评论”按钮 -->
136                 <Button
137                     android:id="@+id/btn_pl"
138                     android:layout_width="wrap_content"
139                     android:layout_height="wrap_content"
140                     android:textColor="#3882b8"
141                     android:textSize="15px"
142                     android:text="  评论(31)" />
143             </TableRow>
144             <TableRow
145                 android:layout_width="wrap_content"
146                 android:layout_height="wrap_content"
147                 android:gravity="center">
148                 <!–“刷新”按钮 -->
149                 <Button
150                     android:id="@+id/btn_refresh"
151                     android:layout_width="wrap_content"
152                     android:layout_height="wrap_content"
153                     android:textColor="#3882b8"
154                     android:textSize="15px"
155                     android:text="刷新"  />
156                 <!–“收藏”按钮 -->
157                 <Button
158                     android:id="@+id/btn_sc"
159                     android:layout_width="wrap_content"
160                     android:layout_height="wrap_content"
161                     android:textColor="#3882b8"
162                     android:textSize="15px"
163                     android:text="收藏"  />
164             </TableRow>
165         </TableLayout>
166     </RelativeLayout>
```

16.10　阅读微博功能实现

（1）新建 ViewActivity.java 用来实现阅读微博的功能，代码如下所示，一开始先声明所需要的变量，接着在 onCreate()函数中初始化各个界面的元素。

```
01  public class ViewActivity extends Activity{
02      //保存当前上下文
03      Context mContext;
04      Weibo weibo;
05      //用于显示用户昵称
06      TextView username;
07      //用于显示微博文字内容
08      TextView content;
09      //用于显示用户头像
10      ImageView icon;
11      //“转发”按钮
12      Button btn_zf;
13      //“评论”按钮
14      Button btn_pl;
15      //微博的 id
16      String key;
17      @Override
18      public void onCreate(Bundle savedInstanceState)
19      {
20          super.onCreate(savedInstanceState);
21          setContentView(R.layout.weibo_detail);
22          weibo=Common.weibo;
23          mContext=this;
24          //初始化界面
25          username=(TextView)findViewById(R.id.username);
26          content=(TextView)findViewById(R.id.text);
27          icon=(ImageView)findViewById(R.id.usericon);
28          //“转发”按钮
29          btn_zf=(Button)findViewById(R.id.btn_gz);
30          //“评论”按钮
31          btn_pl=(Button)findViewById(R.id.btn_pl);
32          //“刷新”按钮
33          Button btn_refresh=(Button)findViewById(R.id.btn_refresh);
34          //“收藏”按钮
35          Button btn_sc=(Button)findViewById(R.id.btn_sc);
36          //顶部返回主页面的 ImageView
```

（2）如下所示，在 onCreate 中获取前一个 Activity 通过 Intent 传递过来的参数 key，可以得知当前用户查看的微博的 id，最后调用 view(key)显示微博的详细信息。

```
01  //获取上一个页面传递过来的 key，key 为某一条微博的 id
02  Intent i=this.getIntent();
03  Bundle b=null;
04  if(!i.equals(null)){
05      b=i.getExtras();
06  }
07  if(b!=null && b.containsKey("key")){
08      key = b.getString("key");
```

```
09      //初始化页面
10      view(key);
```

（3）在函数 view()中，通过 getOneWeibo 获取该微博的详细信息，代码如下所示。首先，如 012 行所示，通过函数 getOneWeibo()返回的 JSON 对象取得用户信息对应的 JSON 对象，接着逐一获取用户的昵称、头像等信息。

如果微博包含图片，还要下载对应的图片，这里仍然使用前面我们介绍过的图像加载类 AsyncImageLoader。最后使用函数 showImg()将图像显示出来，在 showImg 中对图像作了一点小小的处理，如果图像宽度超过 300px，则缩放成宽度为 300px 的图像。

```
001 //初始化页面
002 private void view(String id){
003     //根据微博 id 获得微博详细信息
004     JSONObject data=getOneWeibo(mContext,weibo,id);
005     if(data!=null){
006         JSONObject u;
007         String userName="未知";
008         String userIcon=null;
009         String text=null;
010         try {
011             //获得微博对应用户信息
012             u = data.getJSONObject("user");
013             //取得昵称
014             userName=u.getString("screen name");
015             //取得用户头像地址
016             userIcon=u.getString("profile_image_url");
017             Log.e("userIcon", userIcon);
018             Log.e("username","username"+userName);
019             //取得微博内容
020             text=data.getString("text");
021         } catch (JSONException e) {
022             // TODO Auto-generated catch block
023             e.printStackTrace();
024         }
025         //设置用户昵称
026         username.setText(userName);
027         //设置微博内容
028         content.setText(text);
029         //新建异步图像加载类
030         AsyncImageLoader asyncImageLoader = new AsyncImageLoader();
031         //加载用户头像
032         Drawable cachedImage = asyncImageLoader.loadDrawable(userIcon,
            icon, new ImageCallback(){
033             @Override
034            public void imageLoaded(Drawable imageDrawable,
               ImageView imageView, String imageUrl) {
035                 imageView.setImageDrawable(imageDrawable);
036             }
037         });
038         if (cachedImage == null)
039         {
040             //如果没取到用户头像，则设置成默认的头像
041             icon.setImageResource(R.drawable.default_icon);
042         }
043         else
044         {
045             //设置用户头像
046             icon.setImageDrawable(cachedImage);
```

```
047         }
048         //如果微博带图片则加载图片
049         if(data.has("bmiddle pic")){
050             String picurl=null;
051             String picurl2=null;
052             try {
053                 picurl = data.getString("bmiddle_pic");
054                 picurl2 = data.getString("original_pic");
055             } catch (JSONException e) {
056                 // TODO Auto-generated catch block
057                 e.printStackTrace();
058             }
059             //同样使用异步图像加载类进行加载
060             ImageView pic=(ImageView)findViewById(R.id.pic);
061             Drawable cachedImage2 = asyncImageLoader.loadDrawable
                (picurl,pic, new ImageCallback(){
062                 @Override
063                 public void imageLoaded(Drawable imageDrawable,
                    ImageView imageView, String imageUrl) {
064                     showImg(imageView,imageDrawable);
065                 }
066             });
067             if (cachedImage2 == null)
068             {
069                 //如果没加载到对应图像，则用默认图像代替
070                 pic.setImageResource(R.drawable.default icon);
071             }
072             else
073             {
074                 //设置图像
075                 showImg(pic,cachedImage2);
076             }
077         }
078         String rt=null;
079         String comments=null;
080         try {
081             //取得转发数量
082             rt = data.getString("reposts count");
083             //取得评论数量
084             comments=data.getString("comments_count");
085         } catch (JSONException e) {
086             // TODO Auto-generated catch block
087             e.printStackTrace();
088         }
089         btn_zf.setText(" 转发("+rt+")");
090         btn_pl.setText(" 评论("+comments+")");
091     }
092 }
093 //调用 API 取得对应的微博详细信息
094 public JSONObject getOneWeibo(Context mContext,Weibo weibo,String id)
095 {
096       String url=Weibo.SERVER+"statuses/show.json";
097       WeiboParameters bundle=new WeiboParameters();
098       bundle.add("source", ConstParam.CONSUMER KEY);
099       //传入微博 id
100       bundle.add("id", id);
101       String rlt="";
102         try {
103             rlt = weibo.request(mContext, url, bundle, "GET",
                weibo.getAccessToken());
```

```
104             return new JSONObject(rlt);
105         } catch (Exception e) {
106             // TODO Auto-generated catch block
107             e.printStackTrace();
108             return null;
109         }
110 }
111     //显示微博的图像
112 private void showImg(ImageView view,Drawable img){
113     int w=img.getIntrinsicWidth();
114     int h=img.getIntrinsicHeight();
115     Log.e("w", w+"/"+h);
116     //如果宽度超过 300，则进行缩放处理
117     if(w>300)
118     {
119         int hh=300*h/w;
120         Log.e("hh", hh+"");
121         LayoutParams para=view.getLayoutParams();
122         para.width=300;
123         para.height=hh;
124         view.setLayoutParams(para);
125     }
126     //显示图像
127     view.setImageDrawable(img);
128 }
```

（4）最后在 onCreate 中还需要为各个按键设置监听器，以下代码 027～053 行实现了“评论”按钮的监听器，当单击“评论”按钮的时候将弹出对话框如图 16.15 所示。用户填写评论内容后单击“提交”按钮即可完成评论，完成之后会在最下方显示“评论成功”的 Toast。

同样，转发也是同样的设置方式，只是调用的函数换成了 repost。“收藏”按钮则直接调用收藏对应的 API 进行实现，刷新则调用 view 函数重新进行视图初始化。

```
001 //为“转发”按钮设置按键监听器
002     btn zf.setOnClickListener(new OnClickListener(){
003         @Override
004         public void onClick(View v) {
005             // TODO Auto-generated method stub
006             //膨胀出 EditText 视图
007             LayoutInflater inflater=(LayoutInflater)mContext.
                getSystemService(LAYOUT_INFLATER_SERVICE);
008             final View view=inflater.inflate(R.layout.comment, null);
009             //取得 EditText
010            final EditText retrans=(EditText)view. findViewById(R.id.
               comment);
011             new AlertDialog.Builder(mContext)
012             .setTitle("转发")
013             .setView(view)
014             .setPositiveButton("提交", new android.content.
                 DialogInterface.OnClickListener(){
015                 @Override
016                 public void onClick(DialogInterface dialog, int which) {
017                     // TODO Auto-generated method stub
018                     String str=retrans.getText().toString();
019                     //转发
020                     JSONObject data=repost(mContext,weibo,key,str);
021                     if(!data.has("error"))
```

```
022                         Toast.makeText(ViewActivity.this, "转发成功! ",
                            Toast.LENGTH SHORT).show();
023                  }})
024             .setNegativeButton("取消",null).show();
025         }
026     });
027     //为“评论”按钮设置按键监听器
028     btn pl.setOnClickListener(new OnClickListener(){
029         @Override
030         public void onClick(View v) {
031             //TODO Auto-generated method stub
032             LayoutInflater inflater=(LayoutInflater)mContext.
                getSystemService(LAYOUT_INFLATER_SERVICE);
033             final View view=inflater.inflate(R.layout.comment, null);
034             final EditText comment=(EditText)view.
                findViewById(R.id.comment);
035             new AlertDialog.Builder(mContext)
036             .setTitle("评论")
037             .setView(view)
038             .setPositiveButton("提交", new android.content.
                 DialogInterface.OnClickListener(){
039                 @Override
040                 public void onClick(DialogInterface dialog, int which) {
041                     // TODO Auto-generated method stub
042                     String str=comment.getText().toString();
043                     if(str != null)
044                     {
045                         //评论
046                         JSONObject data=comment(mContext,weibo,key,str);
047                         if(!data.has("error"))
048                         Toast.makeText(ViewActivity.this, "评论成功! ",
                            Toast.LENGTH_SHORT).show();
049                     }
050                 }})
051             .setNegativeButton("取消", null).show();
052         }
053     });
054     //为“刷新”按钮设置按键监听器
055     btn_refresh.setOnClickListener(new OnClickListener(){
056         @Override
057         public void onClick(View v) {
058             // TODO Auto-generated method stub
059             view(key);
060         }
061     });
062     //为“收藏”按钮设置按键监听器
063     btn sc.setOnClickListener(new OnClickListener(){
064         @Override
065         public void onClick(View v) {
066             // TODO Auto-generated method stub
067             //收藏
068             JSONObject data=collect(mContext,weibo,key);
069             if(!data.has("error"))
070                 Toast.makeText(ViewActivity.this, "收藏成功! ",
                    Toast.LENGTH SHORT).show();
071         }
072     });
073     //为返回主页面按钮设置按键监听器
074     btn_home.setOnClickListener(new OnClickListener(){
```

```
075         @Override
076         public void onClick(View v) {
077             // TODO Auto-generated method stub
078             Intent it=new Intent(ViewActivity.this,HomeActivity.class);
079             startActivity(it);
080             ViewActivity.this.finish();
081         }
082     });
083         //调用 API 进行转发
084 public JSONObject repost(Context mContext,Weibo weibo,String id,String str)
085 {
086     //转发的 API
087       String url=Weibo.SERVER+"statuses/repost.json";
088       WeiboParameters bundle=new WeiboParameters();
089       bundle.add("source", ConstParam.CONSUMER KEY);
090       bundle.add("id", id);
091       //添加转发附言
092       bundle.add("status",str);
093       String rlt="";
094         try {
095             rlt = weibo.request(mContext, url, bundle, "POST",
                weibo.getAccessToken());
096             return new JSONObject(rlt);
097         } catch (Exception e) {
098             // TODO Auto-generated catch block
099             e.printStackTrace();
100             return null;
101         }
102 }
103 //调用 API 进行收藏微博
104 public JSONObject collect(Context mContext,Weibo weibo,String id)
105 {
106     //收藏的 API
107       String url=Weibo.SERVER+"favorites/create.json";
108       WeiboParameters bundle=new WeiboParameters();
109       bundle.add("source", ConstParam.CONSUMER KEY);
110       //传入微博的 id
111       bundle.add("id", id);
112       String rlt="";
113         try {
114             rlt = weibo.request(mContext, url, bundle, "POST",
                weibo.getAccessToken());
115             return new JSONObject(rlt);
116         } catch (Exception e) {
117             // TODO Auto-generated catch block
118             e.printStackTrace();
119             return null;
120         }
121 }
122 //调用 API 进行评论微博
123 public JSONObject comment(Context mContext,Weibo weibo,String id,
    String comment)
124 {
125       //评论的 API
126       String url=Weibo.SERVER+"comments/create.json";
127       WeiboParameters bundle=new WeiboParameters();
128       bundle.add("source", ConstParam.CONSUMER KEY);
129       //传入微博的 id
130       bundle.add("id", id);
131       //传入评论的内容
```

```
132         bundle.add("comment", comment);
133         String rlt="";
134           try {
135               rlt = weibo.request(mContext, url, bundle, "POST",
                  weibo.getAccessToken());
136               return new JSONObject(rlt);
137           } catch (Exception e) {
138               // TODO Auto-generated catch block
139               e.printStackTrace();
140               return null;
141           }
142 }
```



图 16.15　评论微博

16.11　知识拓展

本章在显示图像的时候用到了回调函数，那么我们就来详细讲解一下 Android 的回调机制吧。关于回调函数，其实就是一个通过函数指针调用的函数。如果你把函数的指针（地址）作为参数传递给另一个函数，当这个指针被用为调用它所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。

不明白？那么我再详细解释一下。假设客户程序 C 调用服务程序 S 中的某个函数 A，然后 S 又在某个时候反过来调用 C 中的某个函数 B，对于 C 来说，这个 B 便叫做回调函数。例如，Win32 下的窗口过程函数就是一个典型的回调函数。一般说来，C 不会自己调用 B，C 提供 B 的目的就是让 S 来调用它，而且是 C 不得不提供。由于 S 并不知道 C 提供的 B 姓甚名谁，所以 S 会约定 B 的接口规范（函数原型），然后由 C 提前通过 S 的一个函数 R 告诉 S 自己将要使用 B 函数，这个过程称为回调函数的注册，R 称为注册函数。Web Service 以及 Java 的 RMI 都用到回调机制，可以访问远程服务器程序。

还是不明白？好吧，我举个通俗一点的例子。某天，我打电话向你请教问题，当然是个难题，你一时想不出解决方法，我又不能拿着电话在那里傻等，于是我们约定：等你想出办法后打手机通知我，这样，我就挂掉电话办其他事情去了。过了 XX 分钟，我的手机响了，你兴高采烈地说问题已经搞定，应该如此这般处理。故事到此结束。这个例子说明了“异步+回调”的编程模式。其中，你后来打手机告诉我结果便是一个“回调”过程；我的手机号码必须在以前告诉你，这便是注册回调函数；我的手机号码应该有效并且手机

能够接收到你的呼叫，这是回调函数必须符合接口规范。

在 Java 中不允许直接操作指针，那么它的回调是如何实现的呢？答案就是通过接口和内部类来实现。Java 方法回调是功能定义和功能实现分享的一种手段，是一种耦合设计思想。作为一种架构，必须有自己的运行环境，并且提供用户的实现接口。

回调函数的使用步骤如下：

（1）定义接口 Callback，包含回调方法 callback()。

（2）在一个类 Caller 中声明一个 Callback 接口对象 mCallback。

（3）在程序中赋予 Caller 对象的接口成员（mCallback） 一个内部类对象如：

```
1  new Callback () {
2      callback () {
3        //函数的具体实现
4        }
```

这样，在需要的时候可用 Caller 对象的 mCallback 接口成员调用 callback()方法完成回调。回调机制在 Android 框架中使用很多，比如 Button 的单击事件，如下所示，模拟了 Button 的使用。

```
01   //a.定义接口
02     public interface OnClickListener {
03     public void OnClick(Button b);
04     //b.定义 Button
05     public class Button {
06     OnClickListener listener;
07     public void click() {
08             listener.OnClick(this);
09         }
10         public void setOnClickListener(OnClickListener listener) {
11         this.listener = listener;
12         }
13     }
14     //c.将接口对象 OnClickListener 赋给 Button 的接口成员
15     public class Activity {
16     public Activity() {
17     }
18     public static void main(String[] args) {
19     Button button = new Button();
20     button.setOnClickListener(new OnClickListener(){
21                 @Override
22                 public void OnClick(Button b) {
23                 System.out.println("clicked");
24         }
25         });
26         button.click(); //user click,System call button.click();
27     }
```

还有我们很熟悉的 Activity 的生命周期中使用的各个函数，如下所示：

```
01  //a.定义接口
02     public interface Activity{
03         public void onCreate();
04.....
05         public void onDestory();
06     }
07     //b. Activity 接口的实现类 MyActivity
```

```
08      //定义一个类实现 Activity 接口
09      public calss MyActivity implements Activity{
10      @Override//实现方法，简单输出
11      public void onCreate(){
12          System.out.println("onCereate");
13      }
14      .....
15      @Override//实现方法，简单输出
16      public void onDestory(){
17              System.out.println("onDestory");
18  }
19  }
20  //c.系统运行环境类 AndroidSystem
21  //系统运行安装类
22  public class AndroidSystem{
23      //定义常量
24
25      public static final int CREATE=1;
26      ....
27      public static final int DESTORY=2;
28      //运行方法
29      public void run(Activity a,int state){
30          switch(state){
31          case CREATE:
32              a.onCreate;
33              break;
34              ....
35              case DESTORY:
36              a.onDestory();
37              break;
38          }
```

以上可以看出，接口（系统框架）是系统提供的，接口的实现是用户实现的，这样可以达到接口统一，实现不同的效果。系统在不同的状态“回调”我们的实现类，来达到接口和实现的分离。

16.12　本 章 小 结

本章介绍了如何利用新浪微博开放平台开发新浪微博客户端，分别从界面设计和功能实现两个方面逐步给大家分析设计的整个过程。需要注意的一点是，读者需要自行申请 App Key 和 App Secret 才能正常运行此程序。本程序作为演示，还有很多功能有待完善，读者可以根据需要自行添加，毕竟新浪微博提供的 API 有上百个，功能还是很多的，但调用方法千篇一律，希望读者可以举一反三。

第 4 篇　Android 影音应用实战案例

- ⏭　第 17 章　MP3 播放器
- ⏭　第 18 章　Android 照相机
- ⏭　第 19 章　视频播放器

第 17 章　MP3 播放器

音乐早已经成为人们生活不可或缺的一部分，因此 MP3 播放器成为了所有智能手机的必备工具，当然 Android 也不例外，本章将要介绍如何开发一款 Android MP3 播放器。

17.1　主界面设计

MP3 播放器除了需要有播放音乐的功能，还需要有一个漂亮的界面，因此界面设计在本程序开发中占有相当大的比重。

17.1.1　主界面概览

如图 17.1 所示，为 MP3 播放器主界面图，从图中我们可以看出整个界面主要采用从上到下的布局方式。最上方显示音乐的名字和歌手的名字，中间显示专辑的图片，下面一个进度条用于显示当前播放的进度，进度条的下面有 3 个并排的按钮用于控制播放的过程，最下方 3 个按钮用于切换不同的界面。

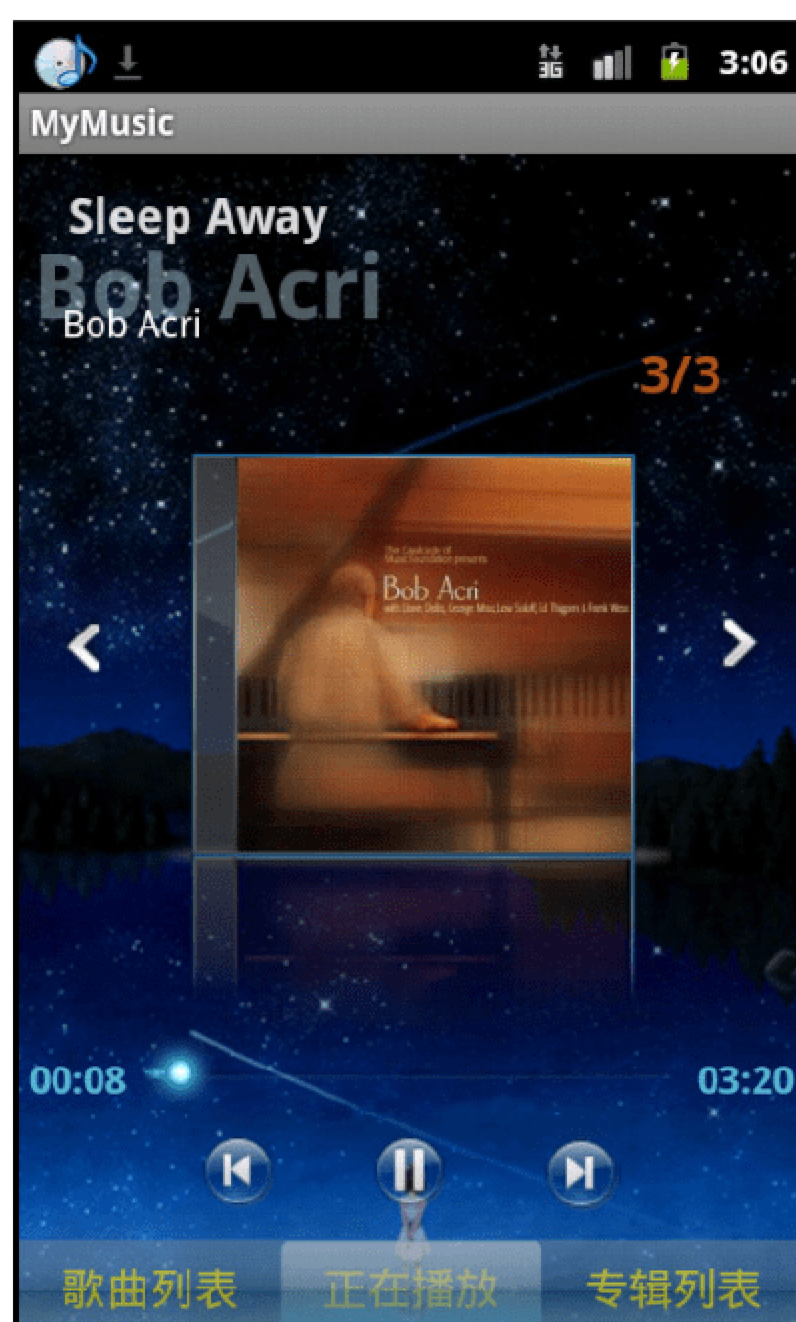


图 17.1　MP3 主界面

在这个界面的实现中，难点在于界面的切换，即中间可滑动区域的实现和底下 3 个界面间切换的实现。

17.1.2　中间切换界面实现

中间的切换界面模仿 Android 桌面 Luncher 的设计，代码如下所示，新建类 DragableLuncher 继承于 ViewGroup。代码 29～59 行为初始化函数部分，其中初始化函数又分为两种，一种是只传入上下文 context 参数，另一种传入上下文和 xml 属性参数。在这两个函数中均用到了变量 mTouchSlop，该变量的作用是用于获取当前触摸屏最小感应距离，当用户移动超过该距离时才触发移动事件。

代码 62～192 行为本类的核心处理函数，用于处理触摸屏移动事件。当用户在触摸屏上滑动时，首先触发的是 onInterceptTouchEvent 事件，接着触发 onTouchEvent 事件。在 onIntercepTouchEvent 函数中主要对触摸事件进行有效性判断，即判断移动的距离是否超过最小记录 mTouchSlop，若满足则返回 true 并接着执行 onTouchEvent 事件，否则返回 false 并且不再执行 onTouchEvent 事件。

在 onTouchEvent()函数中主要对滑动距离进行判断，进行相应的屏幕滚动。主要分为 3 种事件类型，ACTION_DOWN 事件主要记录当前的初始坐标，ACTION_MOVE 事件用于实时响应用户的滑动事件，而 ACTION_UP 事件则根据用户移动的速率，计算出接下去 1s 中可移动的距离，若该距离的绝对值超过 SNAP_VELOCITY，则滚动到旁边的屏幕，否则调用函数 snapToDestination()判断当前移动的距离是否超过屏幕的 1/2，如果超过则移动到下一屏。

代码 248～281 行的函数主要用于界面变化时调用，onLayout 用于将界面的变化传递给其子元素让子元素跟着变化，onMeasure 则将高、宽信息传递给子元素。当我们执行 onTouch()或 invalidate()或 postInvalidate()时都会执行 computeScroll()函数，进行相应的滚动。snapToScreen()和 setToScreen()函数均为切换界面的函数，只是前者带动画，后者不带动画。

```
001 /**
002  * 自定义 View，模仿 Android 桌面 Luncher
003  */
004 public class DragableLuncher extends ViewGroup {
005     //按钮背景色
006     int choseColor, defaultColor;
007     //底部按钮数组
008     ImageButton[] bottomBar;
009     //负责得到滚动属性的对象
010     private Scroller mScroller;
011     //负责触摸的功能类
012     private VelocityTracker mVelocityTracker;
013     //滚动的起始 X 坐标
014     private int mScrollX = 0;
015     //默认显示第几屏
016     private int mCurrentScreen = 0;
017     //滚动的结束 X 坐标
```

```
018     private float mLastMotionX;
019
020     private static final int SNAP_VELOCITY = 1000;
021
022     private final static int TOUCH_STATE_REST = 0;
023     private final static int TOUCH_STATE_SCROLLING = 1;
024
025     private int mTouchState = TOUCH_STATE_REST;
026     //用户滑动距离的最小值
027     private int mTouchSlop = 0;
028
029     public DragableLuncher(Context context) {
030         super(context);
031         mScroller = new Scroller(context);
032         //得到用于判定用户是否滑动的距离的临界值
033         mTouchSlop = ViewConfiguration.get(getContext()).getScaled
            TouchSlop();
034
035         this.setLayoutParams(new ViewGroup.LayoutParams(
036                 ViewGroup.LayoutParams.WRAP_CONTENT,
037                 ViewGroup.LayoutParams.FILL_PARENT));
038     }
039
040     public DragableLuncher(Context context, AttributeSet attrs) {
041         super(context, attrs);
042         mScroller = new Scroller(context);
043
044         mTouchSlop = ViewConfiguration.get(getContext()).getScaled
           TouchSlop();
045
046         this.setLayoutParams(new ViewGroup.LayoutParams(
047                 ViewGroup.LayoutParams.WRAP_CONTENT,
048
049                 ViewGroup.LayoutParams.FILL_PARENT));
050
051         /* 获得 xml 中设置的属性值，这里是指默认显示第几屏幕 */
052         //mCurrentScreen =
053         //attrs.getAttributeResourceValue("http://schemas.android.
           com/apk/res/com.luncher.demo",
054         //"default_screen", 0);
055         TypedArray a = getContext().obtainStyledAttributes(attrs,
056                 R.styleable.DragableLuncher);
057         mCurrentScreen = a.getInteger(
058                 R.styleable.DragableLuncher_default_screen, 0);
059     }
060     //拦截 touch 事件，返回 true 继续执行 ontouch 回调函数
061     @Override
062     public boolean onInterceptTouchEvent(MotionEvent ev) {
063
064         final int action = ev.getAction();
065         if ((action == MotionEvent.ACTION_MOVE)
066                 && (mTouchState != TOUCH_STATE_REST)) {
067             return true;
068         }
069         //取得当前的 x 坐标
070         final float x = ev.getX();
071
072         switch (action) {
073         case MotionEvent.ACTION_MOVE:
```

```
074             //取绝对值
075             final int xDiff = (int) Math.abs(x - mLastMotionX);
076             //若移动的距离小于最小距离则将移动的标志位置为 true，否则置为 false
077             boolean xMoved = xDiff > mTouchSlop;
078             if (xMoved) {
079                 //如果用户沿着 x 轴滑动足够的距离就滚动
080                 mTouchState = TOUCH_STATE_SCROLLING;
081             }
082             break;
083             //触摸到 touch 瞬间记录下 x 坐标
084         case MotionEvent.ACTION_DOWN:
085             //记录滑动的初始位置
086             mLastMotionX = x;
087             //如果停止拖动
088             mTouchState = mScroller.isFinished() ? TOUCH_STATE_REST
089                     : TOUCH_STATE_SCROLLING;
090             break;
091 
092         case MotionEvent.ACTION_CANCEL:
093         case MotionEvent.ACTION_UP:
094             //停止拖动
095             mTouchState = TOUCH_STATE_REST;
096             break;
097         }
098         return mTouchState != TOUCH_STATE_REST;
099     }
100 
101     /**
102      * 设置是否打开触摸滑动
103      */
104     public boolean isOpenTouchAnima(boolean b) {
105         isOpen = b;
106         return isOpen;
107     }
108     //默认触摸滑动打开
109     public boolean isOpen = true;
110     //响应滑动时间
111     @Override
112     public boolean onTouchEvent(MotionEvent event) {
113         if (isOpen) {
114             if (mVelocityTracker == null) {
115                 //获得速率探测器
116                 mVelocityTracker = VelocityTracker.obtain();
117             }
118             //将 touch 事件添加进探测器中
119             mVelocityTracker.addMovement(event);
120             //取得 touch 事件的类型
121             final int action = event.getAction();
122             //取得 x 坐标
123             final float x = event.getX();
124             //处理各种 touch 事件
125             switch (action) {
126             case MotionEvent.ACTION_DOWN:
127                 if (!mScroller.isFinished()) {
128                     mScroller.abortAnimation();
129                 }
130                 //记录下初始位置
131                 mLastMotionX = x;
```

```
132                 break;
133             case MotionEvent.ACTION_MOVE:
134                 final int deltaX = (int) (mLastMotionX - x);
135                 mLastMotionX = x;
136                 if (deltaX < 0) {
137                     if (mScrollX > 0) {
138                         scrollBy(Math.max(-mScrollX, deltaX), 0);
139                     }
140                 } else if (deltaX > 0) {
141                     //取得可滚动的最大距离
142                     final int availableToScroll = getChildAt(
143                             getChildCount() - 1).getRight()
144                             - mScrollX - getWidth();
145                     if (availableToScroll > 0) {
146                         scrollBy(Math.min(availableToScroll, deltaX), 0);
147                     }
148                 }
149                 break;
150             case MotionEvent.ACTION_UP:
151                 //计算当前速率
152                 final VelocityTracker velocityTracker = mVelocityTracker;
153                 velocityTracker.computeCurrentVelocity(1000);
154                 int velocityX = (int) velocityTracker.getXVelocity();
155 
156                 if (velocityX > SNAP_VELOCITY && mCurrentScreen > 0) {
157                     //滑动到左边的界面
158                     snapToScreen(mCurrentScreen - 1);
159                 } else if (velocityX < -SNAP_VELOCITY
160                         && mCurrentScreen < getChildCount() - 1) {
161                     //滑动到右边的界面
162                     snapToScreen(mCurrentScreen + 1);
163                 } else {
164                     //滑动到判定的界面
165                     snapToDestination();
166                 }
167 
168                 if (mVelocityTracker != null) {
169                     mVelocityTracker.recycle();
170                     mVelocityTracker = null;
171                 }
172                 mTouchState = TOUCH_STATE_REST;
173                 break;
174             case MotionEvent.ACTION_CANCEL:
175                 mTouchState = TOUCH_STATE_REST;
176             }
177             mScrollX = this.getScrollX();
178         } else {
179             return false;
180         }
181         if (bottomBar != null) {
182             for (int k = 0; k < bottomBar.length; k++) {
183                 if (k == mCurrentScreen) {
184                     bottomBar[k].setBackgroundColor(choseColor);
185                 } else {
186                     bottomBar[k].setBackgroundColor(defaultColor);
187                 }
188             }
189         }
190 
```

```
191            return true;
192        }
193
194        public void setBottomBarBg(ImageButton[] ib, int choseColor,
195                int defaultColor) {
196            this.bottomBar = ib;
197            this.choseColor = choseColor;
198            this.defaultColor = defaultColor;
199        }
200        //滑动到判定的界面
201        private void snapToDestination() {
202            final int screenWidth = getWidth();
203            final int whichScreen = (mScrollX + (screenWidth / 2))
               / screenWidth;
204            snapToScreen(whichScreen);
205        }
206
207        /**
208         * 带动画效果显示界面
209         */
210        public void snapToScreen(int whichScreen) {
211            mCurrentScreen = whichScreen;
212            final int newX = whichScreen * getWidth();
213            final int delta = newX - mScrollX;
214            mScroller.startScroll(mScrollX, 0, delta, 0, Math.abs(delta)
               * 2);
215            invalidate();
216        }
217
218        /**
219         * 不带动画效果显示界面
220         */
221        public void setToScreen(int whichScreen) {
222            // Log.i(LOG TAG, "set To Screen " + whichScreen);
223            mCurrentScreen = whichScreen;
224            final int newX = whichScreen * getWidth();
225            mScroller.startScroll(newX, 0, 0, 0, 10);
226            invalidate();
227        }
228        //获得当前屏幕是第几屏
229        public int getCurrentScreen() {
230            return mCurrentScreen;
231        }
232        //当主界面布局改变时调用
233        @Override
234        protected void onLayout(boolean changed, int l, int t, int r,
          int b) {
235            int childLeft = 0;
236            //获得子元素的个数
237            final int count = getChildCount();
238            for (int i = 0; i < count; i++) {
239                final View child = getChildAt(i);
240                if (child.getVisibility() != View.GONE) {
241                    final int childWidth = child.getMeasuredWidth();
242                    child.layout(childLeft, 0, childLeft + childWidth,
243                            child.getMeasuredHeight());
244                    childLeft += childWidth;
245                }
```

```
246             }
247         }
248         //取得测量得到的高、宽
249         @Override
250         protected void onMeasure(int widthMeasureSpec,int heightMeasureSpec
            ) {
251             super.onMeasure(widthMeasureSpec, heightMeasureSpec);
252             //提取出宽度
253             final int width = MeasureSpec.getSize(widthMeasureSpec);
254             //提取出宽度的模式
255             final int widthMode = MeasureSpec.getMode(widthMeasureSpec);
256             if (widthMode != MeasureSpec.EXACTLY) {
257                 throw new IllegalStateException("error mode.");
258             }
259             //提取高度的模式
260             final int heightMode = MeasureSpec.getMode(heightMeasureSpec);
261             if (heightMode != MeasureSpec.EXACTLY) {
262                 throw new IllegalStateException("error mode.");
263             }
264 
265             //子元素将被分配给同样的高和宽
266             final int count = getChildCount();
267             for (int i = 0; i < count; i++) {
268                 getChildAt(i).measure(widthMeasureSpec,
                     heightMeasureSpec);
269             }
270             //滚动到指定的屏幕
271             scrollTo(mCurrentScreen * width, 0);
272         }
273         //计算滚动的坐标
274         @Override
275         public void computeScroll() {
276             if (mScroller.computeScrollOffset()) {
277                 mScrollX = mScroller.getCurrX();
278                 scrollTo(mScrollX, 0);
279                 postInvalidate();
280             }
281         }
282     }
```

17.1.3　底部切换界面实现

与中间滚动界面类似的是底部切换界面，通过单击最底下 3 个按钮，可以在“歌曲列表”、“正在播放”和“专辑列表”之间切换，只不过这种切换不带滑动效果。代码如下所示，新建类 BigDragableLuncher，这个类的所有函数均在类 DragableLuncher 中出现过，功能也一样。使用这个类的主要目的是为了将 3 个界面放到该类下，并通过该类的函数实现这 3 个界面的切换。

```
001 /**
002  * 自定义 View，模仿 Android 桌面 Luncher
003  */
004 public class BigDragableLuncher extends ViewGroup {
005 
006     //按钮背景色
007     int choseColor, defaultColor;
```

```
008     //底部按钮数组
009     ImageButton[] bottomBar;
010     //负责得到滚动属性的对象
011     private Scroller mScroller;
012     //滚动的起始 X 坐标
013     private int mScrollX = 0;
014     //默认显示第几屏
015     private int mCurrentScreen = 0;
016
017     public int mTouchSlop = 0;
018
019     public BigDragableLuncher(Context context) {
020         super(context);
021         mScroller = new Scroller(context);
022         //得到状态位
023         mTouchSlop = ViewConfiguration.get(getContext()).getScaled
           TouchSlop();
024         //设置布局参数
025         this.setLayoutParams(new ViewGroup.LayoutParams(
026                 ViewGroup.LayoutParams.WRAP_CONTENT,
027                 ViewGroup.LayoutParams.FILL_PARENT));
028     }
029
030     public BigDragableLuncher(Context context, AttributeSet attrs) {
031         super(context, attrs);
032         mScroller = new Scroller(context);
033
034         mTouchSlop = ViewConfiguration.get(getContext()).getScaled
            TouchSlop();
035         //设置布局参数
036         this.setLayoutParams(new ViewGroup.LayoutParams(
037                 ViewGroup.LayoutParams.WRAP_CONTENT,
038                 ViewGroup.LayoutParams.FILL_PARENT));
039
040         TypedArray a = getContext().obtainStyledAttributes(attrs,
041                 R.styleable.DragableLuncher);
042         mCurrentScreen = a.getInteger(
043                 R.styleable.DragableLuncher_default_screen, 0);
044     }
045     //设置底部按钮颜色
046     public void setBottomBarBg(ImageButton[] ib, int choseColor,
047             int defaultColor) {
048         this.bottomBar = ib;
049         this.choseColor = choseColor;
050         this.defaultColor = defaultColor;
051     }
052     //屏幕滚动
053     public void snapToDestination() {
054         final int screenWidth = getWidth();
055         //滑动距离超过 1/2 屏幕时，进入下一个界面
056         final int whichScreen = (mScrollX + (screenWidth / 2))
            / screenWidth;
057         snapToScreen(whichScreen);
058     }
059
060     /**
061      * 带动画效果显示界面
062      */
```

```
063     public void snapToScreen(int whichScreen) {
064         mCurrentScreen = whichScreen;
065         final int newX = whichScreen * getWidth();
066         final int delta = newX - mScrollX;
067         mScroller.startScroll(mScrollX, 0, delta, 0, Math.abs(delta)
            * 2);
068         invalidate();
069     }
070
071     /**
072      * 不带动画效果显示界面
073      */
074     public void setToScreen(int whichScreen) {
075         mCurrentScreen = whichScreen;
076         final int newX = whichScreen * getWidth();
077         mScroller.startScroll(newX, 0, 0, 0, 10);
078         invalidate();
079     }
080     //取得当前屏幕位置
081     public int getCurrentScreen() {
082         return mCurrentScreen;
083     }
084     //改变布局时调用
085     @Override
086     protected void onLayout(boolean changed, int l, int t, int r,
         int b) {
087         int childLeft = 0;
088
089         final int count = getChildCount();
090         for (int i = 0; i < count; i++) {
091             final View child = getChildAt(i);
092             //从左到右依次排列子元素
093             if (child.getVisibility() != View.GONE) {
094                 final int childWidth = child.getMeasuredWidth();
095                 child.layout(childLeft, 0, childLeft + childWidth,
096                         child.getMeasuredHeight());
097                 childLeft += childWidth;
098             }
099         }
100
101     }
102     //传递高、宽信息
103     @Override
104     protected void onMeasure(int widthMeasureSpec,int heightMeasureSpec
        ) {
105         super.onMeasure(widthMeasureSpec, heightMeasureSpec);
106         //取得宽度值
107         final int width = MeasureSpec.getSize(widthMeasureSpec);
108         final int widthMode = MeasureSpec.getMode(widthMeasureSpec);
109         if (widthMode != MeasureSpec.EXACTLY) {
110             throw new IllegalStateException("error mode.");
111         }
112         final int heightMode = MeasureSpec.getMode(heightMeasureSpec);
113         if (heightMode != MeasureSpec.EXACTLY) {
114             throw new IllegalStateException("error mode.");
115         }
116         //将高宽信息传递给子元素
117         final int count = getChildCount();
```

```
118         for (int i = 0; i < count; i++) {
119             getChildAt(i).measure(widthMeasureSpec, height
                MeasureSpec);
120         }
121         scrollTo(mCurrentScreen * width, 0);
122     }
123     //计算滚动距离
124     @Override
125     public void computeScroll() {
126         if (mScroller.computeScrollOffset()) {
127             mScrollX = mScroller.getCurrX();
128             scrollTo(mScrollX, 0);
129             postInvalidate();
130         }
131     }
132 }
```

17.1.4　主界面结构布局

从前面的分析我们知道，整个界面采用模仿 Android Luncher 的设计，也就是将多个界面包含在一个视图组中，再通过滑动或者按键在这些子元素界面间来回切换。

为了达到比较精确的排列，整个布局大多采用 RelativeLayout 的方式，代码如下所示。这个界面首先分成两大部分，一部分是由 BigDragableLuncher 构成的界面主体，一部分是下方用于切换界面的 3 个按钮。

在 BigDragableLuncher 中界面从上到下分为 3 部分，最上面是显示歌曲名称、歌手名字、专辑名称等简要信息。中间部分则由一个 DragableLuncher 组成，这个 DragableLuncher 与 BigDragableLuncher 功能类似，都是用于将多个界面放到一起，并控制这些界面的切换。下面部分则是歌曲的进度条以及控制歌曲播放的 3 个按钮。

```
001 <?xml version="1.0" encoding="utf-8"?>
002 <RelativeLayout
003     xmlns:android="http://schemas.android.com/apk/res/android"
004     android:orientation="vertical"
005     android:layout_width="fill_parent"
006     android:layout_height="fill_parent"
007     android:background="@drawable/musicplayer_bkg"
008     android:gravity="center"
009     android:layout_gravity="top">
010    <!-- 类似于 Luncher，用于切换内部子界面 -->
011     <com.android.supermario.BigDragableLuncher
012         xmlns:android="http://schemas.android.com/apk/res/android"
013         xmlns:guojs="http://schemas.android.com/apk/res/com.android.
        supermario"
014         android:id="@+id/all_space"
015         android:layout_width="fill_parent"
016         android:layout_height="fill_parent"
017         guojs:default_screen="1">
018         <!-- 第一个界面，音乐列表 -->
019         <include android:id="@+id/music_list" layout="@layout
            /musiclist"/>
020         <!-- 主界面根元素 -->
021      <RelativeLayout
```

```
022        android:id="@+id/linearlayout1"
023        android:orientation="vertical"
024        android:layout_width="fill_parent"
025        android:layout_height="wrap_content">
026        <!-- 上面部分，用于显示歌曲名称、歌手名字、歌曲序号和专辑名字 -->
027        <RelativeLayout
028            android:id="@+id/linearlayout2"
029            android:orientation="vertical"
030            android:layout_width="fill_parent"
031            android:layout_height="100dip">
032            <!-- 音乐名称 -->
033            <TextView
034                android:id="@+id/music_name"
035                android:text="无歌曲播放"
036                android:textSize="19sp"
037                android:textStyle="bold"
038                android:textColor="#ddffffff"
039                android:layout_width="wrap_content"
040                android:layout_height="wrap_content"
041                android:layout_marginTop="11dip"
042                android:layout_marginLeft="20dip"
043                android:singleLine="true"/>
044            <!-- 音乐专辑名称 -->
045            <TextView
046                android:id="@+id/music_album"
047                android:textSize="36sp"
048                android:textColor="#66ceedf9"
049                android:textStyle="bold"
050                android:layout_width="300dip"
051                android:layout_height="wrap_content"
052                android:layout_marginTop="28dip"
053                android:layout_marginLeft="5dip"
054                android:singleLine="true"
055                android:focusable="true"
056                android:focusableInTouchMode="true"
057                android:ellipsize="marquee"
058                android:marqueeRepeatLimit="marquee_forever"/>
059            <!-- 歌手名字 -->
060            <TextView
061                android:id="@+id/music_artist"
062                android:textSize="15sp"
063                android:textColor="#ffffff"
064                android:layout_width="wrap_content"
065                android:layout_height="wrap_content"
066                android:layout_marginTop="58dip"
067                android:layout_marginLeft="17dip"/>
068            <!-- 歌曲序号 -->
069            <TextView
070                android:id="@+id/music_number"
071                android:text="0/0"
072                android:textSize="22sp"
073                android:textStyle="bold"
074                android:textColor="#bbf3731e"
075                android:layout_width="60dip"
076                android:layout_height="wrap_content"
077                android:layout_marginTop="73dip"
078                android:layout_marginRight="20dip"
079                android:layout_alignParentRight="true"
080                android:gravity="center_horizontal"/>
```

```
081             </RelativeLayout>
082         <!--中间可滚动部分  -->
083         <RelativeLayout
084             android:id="@+id/relativelayout1"
085             android:layout_width="fill_parent"
086             android:layout_height="260dip"
087             android:layout_below="@id/linearlayout2">
088         <!-- 仿 Luncher 设计，可左右滑动进行 3 个界面的切换 -->
089         <com.android.supermario.DragableLuncher xmlns:android="http:
            //schemas.android.com/apk/res/android"
090                 xmlns:guojs="http://schemas.android.com/apk/res
                /com.android.supermario"
091                 android:id="@+id/space"
092                 android:layout_width="fill_parent"
093                 android:layout_height="fill_parent"
094                 guojs:default_screen="1">
095                  <!-- 左边显示动画界面 -->
096                 <include android:id="@+id/left" layout="@layout/
                 left_mediaview"/>
097                 <!-- 中间显示专辑图片 -->
098                 <include android:id="@+id/center" layout="@layout/
                 center_special"/>
099                 <!-- 右边显示歌词 -->
100                 <include android:id="@+id/right" layout="@layout/
                 right_lrc"/>
101         </com.android.supermario.DragableLuncher>
102         </RelativeLayout>
103         <!-- 最下面部分，用于显示歌曲进度和控制歌曲播放的按钮 -->
104           <RelativeLayout
105             android:id="@+id/relativelayout2"
106             android:layout_width="fill_parent"
107             android:layout_height="72dip"
108             android:layout_below="@+id/relativelayout1">
109             <!-- 歌曲进度显示 -->
110             <LinearLayout
111                 android:id="@+id/linearLayout3"
112                 android:layout_width="fill_parent"
113                 android:layout_height="wrap_content"
114                 android:gravity="center">
115                 <!--当前播放时间 -->
116                 <TextView
117                     android:id="@+id/time_tv1"
118                     android:text=" 00:00 "
119                     android:textSize="16sp"
120                     android:textStyle="bold"
121                     android:textColor="#bb7af9fe"
122                     android:layout_width="35dip"
123                     android:layout_height="18dip"
124                     android:layout_weight="1"
125                     android:gravity="left"/>
126                 <!-- 进度条，显示播放进度 -->
127                 <SeekBar
128                     android:id="@+id/player_seekbar"
129                     android:layout_width="220dip"
130                     android:layout_height="wrap_content"
131                     android:progressDrawable="@drawable/seekbar_style"
132                     android:background="@drawable/play_progress
                    _background"
```

```
133                     android:thumb="@drawable/thumb"
134                     android:progress="0"
135                     android:max="0"/>
136                 <!-- 显示歌曲总时间 -->
137                 <TextView
138                     android:id="@+id/time_tv2"
139                     android:text=" 00:00 "
140                     android:textSize="16sp"
141                     android:textStyle="bold"
142                     android:textColor="#bb7af9fe"
143                     android:layout width="35dip"
144                     android:layout height="18dip"
145                     android:layout weight="1"
146                     android:gravity="right"/>
147             </LinearLayout>
148             <!-- 控制音乐播放按钮 -->
149            <LinearLayout
150              android:layout width="wrap content"
151              android:layout height="wrap content"
152              android:layout centerHorizontal="true"
153              android:layout alignParentBottom="true">
154                 <!-- 前一首 -->
155                 <ImageButton
156                     android:id="@+id/ib1"
157                     android:src="@drawable/left button"
158                     android:background="#00000000"
159                     android:layout width="wrap content"
160                     android:layout height="wrap content"/>
161                 <!-- 暂停、播放按钮 -->
162                  <ImageButton
163                     android:id="@+id/ib2"
164                     android:background="@drawable/play_button"
165                     android:layout_width="wrap_content"
166                     android:layout_height="wrap_content"
167                     android:layout marginLeft="40dip"/>
168                 <!-- 下一首 -->
169                 <ImageButton
170                     android:id="@+id/ib3"
171                     android:src="@drawable/right button"
172                     android:background="#00000000"
173                     android:layout_width="wrap_content"
174                     android:layout_height="wrap_content"
175                     android:layout_marginLeft="40dip"/>
176             </LinearLayout>
177       </RelativeLayout>
178       </RelativeLayout>
179     <!-- 右边界面，显示专辑列表 -->
180     <include android:id="@+id/grid_special" layout="@layout
        /gridspecial" />
181 </com.android.supermario.BigDragableLuncher>
182     <!-- 底部界面布局部分，用于显示歌曲列表、正在播放、专辑列表界面切换按钮 -->
183     <LinearLayout
184         android:orientation="horizontal"
185         android:layout width="fill parent"
186         android:layout_height="40dip"
187         android:layout_alignParentBottom="true">
188         <!—“歌曲列表”按钮 -->
189         <Button
190             android:id="@+id/Button1"
```

```
191             android:text="歌曲列表"
192             android:textSize="18sp"
193             android:textStyle="bold"
194             android:textColor="#fafa2d"
195             android:background="#00000000"
196             android:layout_width="wrap_content"
197             android:layout_height="match_parent"
198             android:layout_weight="1"></Button>
199         <!–“正在播放”按钮 -->
200         <Button
201             android:id="@+id/Button2"
202             android:text="正在播放"
203             android:textSize="18sp"
204             android:textStyle="bold"
205             android:textColor="#fafa2d"
206             android:background="#00000000"
207             android:layout_width="wrap_content"
208             android:layout_height="match_parent"
209             android:layout_weight="1"></Button>
210         <!–“专辑列表”按钮 -->
211         <Button
212             android:id="@+id/Button3"
213             android:text="专辑列表"
214             android:textSize="18sp"
215             android:textStyle="bold"
216             android:textColor="#fafa2d"
217             android:background="#00000000"
218             android:layout_width="wrap_content"
219             android:layout_height="match_parent"
220             android:layout_weight="1"></Button>
221     </LinearLayout>
222     <!-- 与上面的 button 位置重合，为 button 提供背景图片 -->
223     <LinearLayout
224         android:orientation="horizontal"
225         android:layout width="fill parent"
226         android:layout_height="40dip"
227         android:layout_alignParentBottom="true">
228         <!-- 歌曲播放按钮 -->
229         <ImageButton
230             android:id="@+id/imageButton1"
231             android:src="@drawable/big button style"
232             android:background="#00000000"
233             android:layout width="wrap content"
234             android:layout height="wrap content"
235             android:layout weight="1"></ImageButton>
236         <!–“正在播放”按钮 -->
237         <ImageButton
238             android:id="@+id/imageButton2"
239             android:src="@drawable/big_button_style"
240             android:background="#00000000"
241             android:layout_width="wrap_content"
242             android:layout height="wrap content"
243             android:layout weight="1"></ImageButton>
244         <!–“专辑列表”按钮 -->
245         <ImageButton
246             android:id="@+id/imageButton3"
247             android:src="@drawable/big button style"
248             android:background="#00000000"
249             android:layout_width="wrap_content"
```

```
250             android:layout height="wrap content"
251             android:layout weight="1"></ImageButton>
252     </LinearLayout>
253 </RelativeLayout>
```

17.2　左右界面设计

17.2.1　歌曲列表界面布局

歌曲列表界面在主界面的左边，如图 17.2 所示，布局代码如下所示。整个界面只有两个元素，一个是用 TextView 显示的标题，一个是主体的 ListView 用于显示音乐列表。

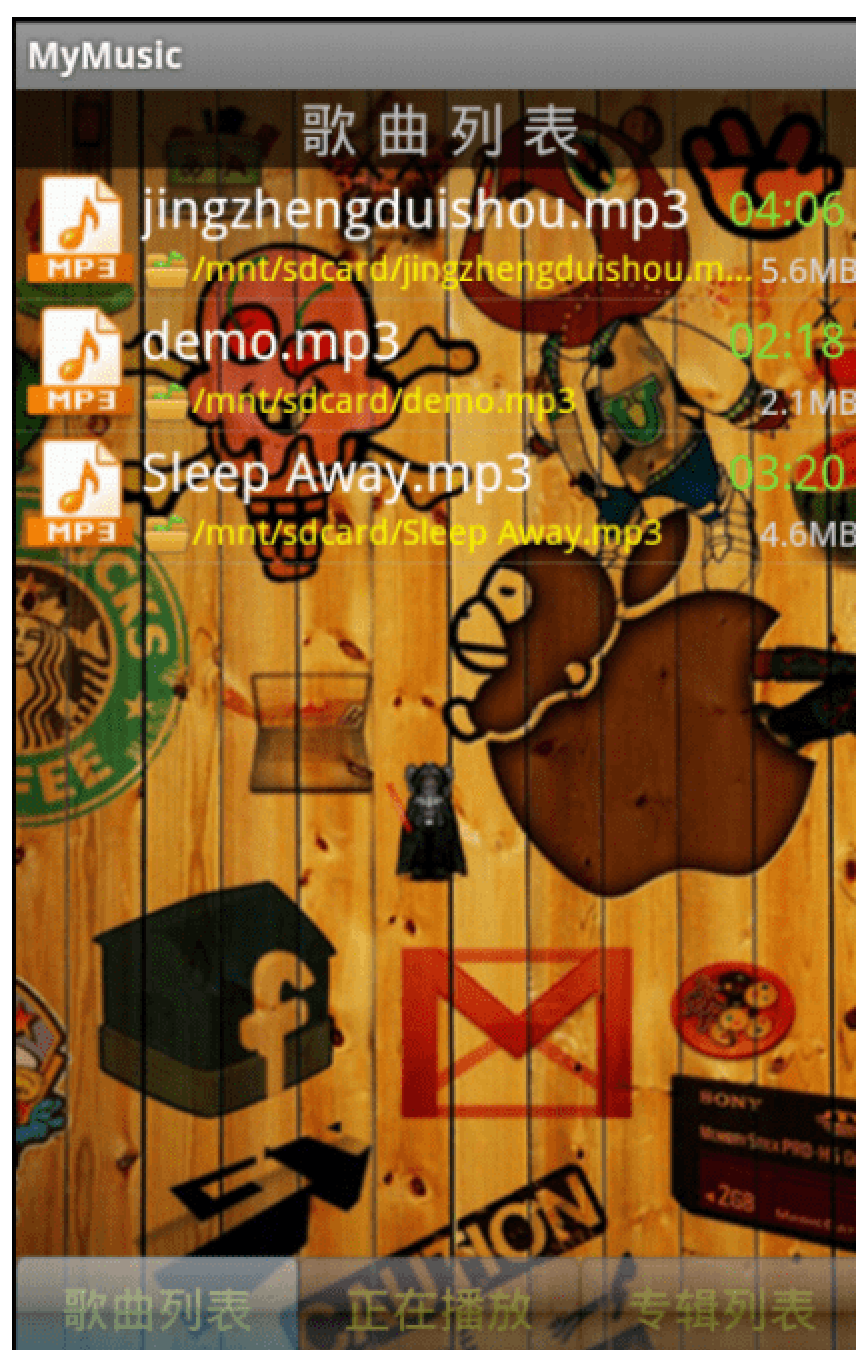


图 17.2　歌曲列表

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayoutxmlns:android="http://schemas.android.com/apk/res/
    android"
03      android:orientation="vertical"
04      android:layout_width="fill_parent"
05      android:layout_height="fill_parent"
06      android:background="@drawable/musiclist bkg">
07      <!-- 标题 -->
08      <TextView
09          android:layout width="fill parent"
10          android:layout height="30dip"
11          android:text="歌 曲 列 表"
12          android:textSize="22sp"
13          android:background="#a0000000"
```

```
14          android:gravity="center"/>
15      <!-- 音乐列表 -->
16      <ListView
17          android:id="@+id/listView1"
18          android:layout_width="fill_parent"
19          android:layout_height="415dip"
20          android:cacheColorHint="#00000000"></ListView>
21  </LinearLayout>
```

歌曲播放列表的每一行元素布局代码如下所示，整个界面采用从左到右的布局。最左边显示音乐的 ICON；中间部分分为上面和下面，上面显示歌曲名字，下面显示歌曲时间；右边部分的上半部显示歌曲的路径，下半部显示歌曲的文件大小。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.
     com/apk/res/android"
03      android:orientation="horizontal"
04      android:layout_width="match_parent"
05      android:layout_height="match_parent">
06      <RelativeLayout
07          android:layout_width="match_parent"
08          android:layout_height="wrap_content"
09          android:padding="1dip">
10      <!-- 歌曲的图标 -->
11      <ImageView
12          android:id="@+id/iv1"
13          android:src="@drawable/ic_mp3"
14          android:layout_width="46dip"
15          android:layout_height="46dip"></ImageView>
16          <RelativeLayout
17              android:id="@+id/relativelayout1"
18              android:layout_width="fill_parent"
19              android:layout_height="wrap_content"
20              android:layout_toRightOf="@id/iv1">
21              <!-- 歌曲名字 -->
22              <TextView
23                  android:id="@+id/tv1"
24                  android:text="TextView"
25                  android:textSize="20sp"
26                  android:textColor="#ffffff"
27                  android:layout_width="228dip"
28                  android:layout_height="wrap_content"
29                  android:singleLine="true"></TextView>
30              <!-- 歌曲播放总时间 -->
31              <TextView
32                  android:id="@+id/tv2"
33                  android:text="TextView"
34                  android:textSize="18sp"
35                  android:textColor="#7eeb3d"
36                  android:layout_width="wrap_content"
37                  android:layout_height="wrap_content"
38                  android:layout_centerVertical="true"
39                  android:layout_alignParentRight="true"></TextView>
40          </RelativeLayout>
41
42          <RelativeLayout
43              android:layout_width="fill_parent"
44              android:layout_height="wrap_content"
45              android:layout_toRightOf="@id/iv1"
46              android:layout_below="@id/relativelayout1">
```

```
47              <!-- 文件夹图标，用于标识文件路径 -->
48              <ImageView
49                  android:id="@+id/iv2"
50                  android:src="@drawable/folder"
51                  android:layout_width="19dip"
52                  android:layout_height="19dip"/>
53              <!-- 文件路径 -->
54              <TextView
55                  android:id="@+id/tv3"
56                  android:text="TextView"
57                  android:textSize="13sp"
58                  android:textColor="#ffff00"
59                  android:layout_width="215dip"
60                  android:layout_height="wrap_content"
61                  android:layout_toRightOf="@id/iv2"
62                  android:singleLine="true"></TextView>
63              <!-- 文件大小 -->
64              <TextView
65                  android:id="@+id/tv4"
66                  android:text="TextView"
67                  android:textSize="13sp"
68                  android:textColor="#dadbe2"
69                  android:layout_width="wrap_content"
70                  android:layout_height="wrap_content"
71                  android:layout_centerVertical="true"
72                  android:layout_alignParentRight="true"></TextView>
73          </RelativeLayout>
74      </RelativeLayout>
75  </LinearLayout>
```

17.2.2　专辑列表界面布局

专辑列表与音乐播放列表类似，整个视图仅仅包含一个 GridView 元素，如图 17.3 所示。

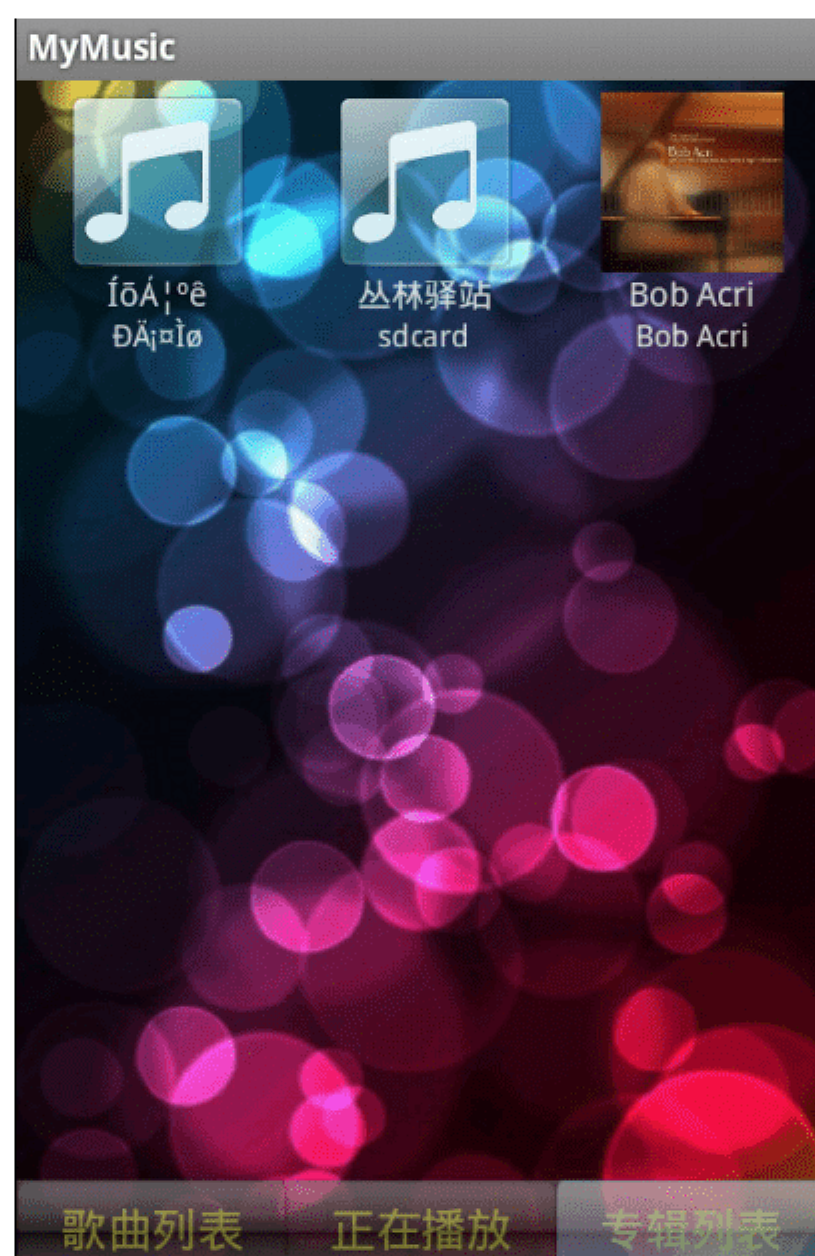


图 17.3　专辑列表

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <RelativeLayout
03   xmlns:android="http://schemas.android.com/apk/res/android"
04   android:orientation="vertical"
05   android:layout_width="match_parent"
06   android:layout_height="match_parent"
07   android:background="@drawable/gridspecial_bkg">
08     <!-- 专辑列表 -->
09   <GridView
10     android:id="@+id/gridview1"
11     android:numColumns="auto_fit"
12     android:layout_width="fill_parent"
13     android:layout_height="445dip"
14     android:columnWidth="90dip"
15     android:horizontalSpacing="10dip"
16     android:verticalSpacing="10dip"
17     android:stretchMode="columnWidth"></GridView>
18 </RelativeLayout>
```

每一个专辑信息显示的界面实现代码如下所示，采用从上到下的布局方式，最上面显示专辑封面，下面依次显示歌手名字和专辑名。

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <LinearLayout
03   xmlns:android="http://schemas.android.com/apk/res/android"
04   android:orientation="vertical"
05   android:layout_width="match_parent"
06   android:layout_height="match_parent">
07       <RelativeLayout
08         android:id="@+id/rl1"
09         android:layout_width="100dip"
10         android:layout_height="115dip">
11             <!-- 专辑封面 -->
12           <ImageView
13             android:id="@+id/gridspecial_view1"
14             android:background="@drawable/default_album"
15             android:layout_width="72dip"
16             android:layout_height="72dip"
17             android:layout_centerHorizontal="true"/>
18           <!-- 歌手名字 -->
19           <TextView
20             android:id="@+id/artist_xxx"
21             android:text="歌手"
22             android:textSize="13sp"
23             android:textColor="#cdffffff"
24             android:layout_width="wrap_content"
25             android:layout_height="17dip"
26             android:layout_below="@id/gridspecial_view1"
27             android:layout_centerHorizontal="true"/>
28           <!-- 专辑名 -->
29           <TextView
30             android:id="@+id/album_xxx"
31             android:text="专辑名"
32             android:textSize="12sp"
33             android:textColor="#cdffffff"
34             android:layout_width="wrap_content"
35             android:layout_height="wrap_content"
36             android:layout_below="@id/artist_xxx"
```

```
37              android:layout centerHorizontal="true"
38              android:singleLine="true"/>
39          </RelativeLayout>
40  </LinearLayout>
```

17.3　中间滑动界面

17.3.1　左侧视图——播放动画

左边视图主要用于播放 gif 动画，一般 MP3 播放器都会显示一个根据当前的节奏进行变化的动画，本程序的动画比较粗糙仅作为演示。代码如下所示，是左侧视图的界面布局，由两部分组成，左边是自定义视图类 RunGif，右边是一个指向右侧的箭头图片。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayoutxmlns:android="http://schemasandroid.com/apk/res/
    android"
03      android:orientation="vertical"
04      android:layout width="fill parent"
05      android:layout height="fill parent">
06      <!-- 用于播放 gif 动画 -->
07      <RelativeLayout
08          android:id="@+id/relativelayout1"
09          android:layout width="wrap content"
10          android:layout height="wrap content"
11          android:layout_centerHorizontal="true"
12          android:layout_marginTop="10dip">
13          <!-- 自定义播放动画视图类 -->
14          <com.android.supermario.RunGif
15              android:id="@+id/runGif1"
16              android:layout_width="fill_parent"
17              android:layout_height="240dip" />
18      </RelativeLayout>
19      <!-- 向右箭头 -->
20      <ImageView
21          android:id="@+id/iv2"
22          android:src="@drawable/right arrows"
23          android:layout width="wrap content"
24          android:layout height="wrap content"
25          android:layout_alignParentRight="true"
26          android:layout_marginTop="76dip" />
27  </RelativeLayout>
```

如下所示是类 RunGif 的实现代码，主要通过 Android 自带的 Movie 类来显示 gif 动画。在构造函数中取得动画资源文件 mediaview_gif1，并在 onDraw 中显示。

在函数 onDraw()中首先记录第一帧的播放时间，并获得动画的时长，接着根据当前的时间计算出播放到第几帧，最后显示对应帧的画面。

```
01  package com.android.supermario;
02  import android.content.Context;
03  import android.graphics.Canvas;
```

```
04  import android.graphics.Movie;
05  import android.util.AttributeSet;
06  import android.view.View;
07  //播放动画视图类
08  public class RunGif extends View {
09      public Movie mMovie;
10      public long mMovieStart;
11      //动画播放的标志
12      public static boolean flag = false;
13      //构造函数用于初始化动画
14      public RunGif(Context context) {
15          super(context);
16          // TODO Auto-generated constructor stub
17          mMovie = Movie.decodeStream(getResources().openRawResource(
18                  R.raw.mediaview_gif1));
19      }
20      //是否播放动画的标志
21      public boolean setFlag(boolean b) {
22          flag = b;
23          return flag;
24      }
25      public RunGif(Context context, AttributeSet as) {
26          super(context, as);
27          // TODO Auto-generated constructor stub
28          //带属性的构造函数
29          mMovie = Movie.decodeStream(getResources().openRawResource(
30                  R.raw.mediaview_gif1));
31      }
32      //绘制动画
33      @Override
34      protected void onDraw(Canvas canvas) {
35          // TODO Auto-generated method stub
36          long now = android.os.SystemClock.uptimeMillis();
37          if (flag == true) {
38              if (mMovieStart == 0) {
39                  //播放第一帧
40                  mMovieStart = now;
41              }
42              if (mMovie != null) {
43                  //取出动画的时长
44                  int dur = mMovie.duration();
45                  if (dur == 0) {
46                      dur = 15000;
47                  }
48                  //计算出需要显示第几帧
49                  int relTime = (int) ((now - mMovieStart) % dur);
50                  //设置要显示的帧
51                  mMovie.setTime(relTime);
52                  //显示
53                  mMovie.draw(canvas, 90, 30);
54                  //作用是刷新当前 View
55                  invalidate();
56              }
57          }
58      }
59  }
```

17.3.2　中间视图——显示专辑

如下所示，新建布局文件 center_special.xml 用于显示专辑图片，从前面的效果图我们也可以清楚地看到，整个显示区域包括左右箭头和和中间的专辑图片显示框架。而这个框架又包含背景、倒影和专辑图片本身。

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <RelativeLayout
03   xmlns:android="http://schemas.android.com/apk/res/android"
04   android:orientation="vertical"
05   android:layout_width="match_parent"
06   android:layout_height="match_parent">
07     <!-- 专辑封面显示区 -->
08     <RelativeLayout
09         android:id="@+id/relativelayout1"
10         android:layout_width="wrap_content"
11         android:layout_height="250.6dip"
12         android:layout_centerHorizontal="true"
13         android:layout_marginTop="10dip">
14         <!-- 显示倒影 -->
15         <ImageView
16             android:id="@+id/inverted_view"
17             android:layout_width="160.5dip"
18             android:layout_height="250.6dip"
19             android:layout_marginTop="14.2dip"
20             android:layout_marginLeft="43.7dip"/>
21         <!-- 显示背景图片 -->
22         <ImageView
23             android:id="@+id/music_view"
24             android:src="@drawable/album_bkg"
25             android:layout_width="228.6dip"
26             android:layout_height="250.7dip"/>
27         <!-- 默认专辑封面 -->
28         <ImageView
29             android:id="@+id/music_AlbumArt"
30             android:src="@drawable/album"
31             android:layout_width="160.5dip"
32             android:layout_height="160.5dip"
33             android:layout_marginTop="14.1dip"
34             android:layout_marginLeft="43.8dip"/>
35     </RelativeLayout>
36     <!-- 左箭头 -->
37     <ImageView
38         android:id="@+id/iv1"
39         android:src="@drawable/left_arrows"
40         android:layout_width="wrap_content"
41         android:layout_height="wrap_content"
42         android:layout_alignParentLeft="true"
43         android:layout_marginTop="75dip"/>
44     <!-- 右箭头 -->
45     <ImageView
46         android:id="@+id/iv2"
47         android:src="@drawable/right_arrows"
48         android:layout_width="wrap_content"
49         android:layout_height="wrap_content"
50         android:layout_alignParentRight="true"
```

```
51          android:layout marginTop="75dip"/>
52  </RelativeLayout>
```

17.3.3　右侧视图——显示歌词

如图 17.4 所示，为歌词界面，代码如下所示，在界面布局文件中设置界面左边显示一个向左的箭头，右边显示一个用于显示歌词的自定义视图。



图 17.4　歌词显示界面

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <RelativeLayout
03    xmlns:android="http://schemas.android.com/apk/res/android"
04    android:orientation="vertical"
05    android:layout_width="match_parent"
06    android:layout_height="match_parent">
07      <!-- 左箭头 -->
08      <ImageView
09          android:id="@+id/iv1"
10          android:src="@drawable/left arrows"
11          android:layout width="wrap content"
12          android:layout height="wrap content"
13          android:layout alignParentLeft="true"
14          android:layout marginTop="74dip"/>
15      <!-- 歌词显示区域 -->
16      <com.android.supermario.LrcView
17          android:id="@+id/LyricShow"
```

```
18          android:text="无歌曲播放"
19          android:textColor="#99ffffff"
20          android:layout_width="fill_parent"
21          android:layout_height="230dip"
22          android:layout_marginTop="10dip"
23          android:gravity="center">
24      </com.android.supermario.LrcView>
25  </RelativeLayout>
```

如下所示是歌词显示类，在该类中定义了两种画笔：CurrentPaint 和 NotCurrentPaint 分别用于显示当前播放的歌词和非当前播放的歌词，同时规定了每行位置的高度为 25。在初始化函数中将歌词语句列表传递给类的成员变量 mSentenceEntities，在 onDraw()函数中根据当前歌词播放的位置，高亮显示对应的语句，其他语句则以另一个种格式显示。

```
001 package com.android.supermario;                    //声明包语句
002~11 行为引入相关类，这里不再列举，请阅读光盘内容
//……
012 /**
013  * 自定义绘画歌词，产生滚动效果
014  */
015 public class LrcView extends TextView {
016     //视图的宽度
017     private float width;
018     //视图的高度
019     private float high;
020     //当前播放歌词的画笔
021     private Paint CurrentPaint;
022     //非当前播放歌词的画笔
023     private Paint NotCurrentPaint;
024     //每行文字的高度
025     private float TextHigh = 25;
026     //非当前播放歌词的子图大小
027     private float TextSize = 18;
028     //歌词显示的位置
029     private int Index = 0;
030     //歌词语句列表
031     private List<LrcContent> mSentenceEntities = new ArrayList<Lrc
        Content>();
032     //初始化歌词语句列表
033     public void setSentenceEntities(List<LrcContent> mSentence
        Entities) {
034         this.mSentenceEntities = mSentenceEntities;
035     }
036     //3 种不同参数的初始化
037     public LrcView(Context context) {
038         super(context);
039         // TODO Auto-generated constructor stub
040         init();
041     }
042     public LrcView(Context context, AttributeSet attrs, int
        defStyle) {
043         super(context, attrs, defStyle);
044         // TODO Auto-generated constructor stub
045         init();
046     }
047     public LrcView(Context context, AttributeSet attrs) {
048         super(context, attrs);
```

```
049          // TODO Auto-generated constructor stub
050          init();
051      }
052      //初始化函数
053      private void init() {
054          //TODO Auto-generated method stub
055          setFocusable(true);
056          //高亮部分
057          CurrentPaint = new Paint();
058          CurrentPaint.setAntiAlias(true);
059          CurrentPaint.setTextAlign(Paint.Align.CENTER);
060          //非高亮部分
061          NotCurrentPaint = new Paint();
062          NotCurrentPaint.setAntiAlias(true);
063          NotCurrentPaint.setTextAlign(Paint.Align.CENTER);
064      }
065      //绘制视图
066      @Override
067      protected void onDraw(Canvas canvas) {
068          // TODO Auto-generated method stub
069          super.onDraw(canvas);
070          if (canvas == null) {
071              return;
072          }
073          //设置歌词字体的颜色
074          CurrentPaint.setColor(Color.argb(210, 251, 248, 29));
075          NotCurrentPaint.setColor(Color.argb(140, 255, 255, 255));
076          //设置当前歌词字体和大小
077          CurrentPaint.setTextSize(24);
078          CurrentPaint.setTypeface(Typeface.SERIF);
079          //设置非当前歌词字体和为大小
080          NotCurrentPaint.setTextSize(TextSize);
081          NotCurrentPaint.setTypeface(Typeface.DEFAULT);
082          try {
083              setText("");
084              //显示当前播放的歌词
085              canvas.drawText(mSentenceEntities.get(Index).getLrc()
                 , width / 2,
086                      high / 2, CurrentPaint);
087              float tempY = high / 2;
088              //画出本句之前的句子
089              for (int i = Index - 1; i >= 0; i--) {
090                  //向上推移
091                  tempY = tempY - TextHigh;
092                  //显示歌词
093                  canvas.drawText(mSentenceEntities.get(i).getLrc(),
                     width / 2,
094                          tempY, NotCurrentPaint);
095              }
096              tempY = high / 2;
097              //画出本句之后的句子
098              for (int i = Index + 1; i < mSentenceEntities.size(); i++) {
099                  //往下推移
100                  tempY = tempY + TextHigh;
101                  //显示歌词
102                  canvas.drawText(mSentenceEntities.get(i).getLrc(),
                     width / 2,
103                          tempY, NotCurrentPaint);
104              }
```

```
105         } catch (Exception e) {
106             setText("未找到歌词文件");
107         }
108     }
109     //视图大小改变时
110     @Override
111     protected void onSizeChanged(int w, int h, int oldw, int oldh) {
112         // TODO Auto-generated method stub
113         super.onSizeChanged(w, h, oldw, oldh);
114         this.width = w;
115         this.high = h;
116     }
117     //设置歌词显示的位置
118     public void SetIndex(int index) {
119         this.Index = index;
120     }
121 }
```

与歌词视图类相关的还有歌词进程控制类 LrcProgress，在该类中还有子类 LrcContent 用于存放每一句歌词的内容和时间信息。通过调用 readLRC()函数，可以加载歌曲对应的歌词文件，这里要注意将歌词文件与歌曲文件放在同一个目录下并保持名字一致。

在 readLRC()函数中，将歌词文件的时间和歌词内容进行分离，并分别存储到类 LrcContent 的成员变量 Lrc 和 Lrc_time 中，最后把一个个 LrcContent 类存放到列表 LrcList 中。

```
001 package com.android.supermario;              //声明包语句
002~011 行为引入相关类，这里不再列举，请阅读光盘内容
//……
012 /**
013  * 处理歌词文件的类
014  */
015 public class LrcProcess {
016     //歌词类列表
017     private List<LrcContent> LrcList;
018     //用于存放每一句歌词
019     private LrcContent mLrcContent;
020     //初始化函数
021     public LrcProcess() {
022         //实例化类
023         mLrcContent = new LrcContent();
024         LrcList = new ArrayList<LrcContent>();
025     }
026     /**
027      * 读取歌词文件的内容
028      */
029     public String readLRC(String song path) {
030         StringBuilder stringBuilder = new StringBuilder();
031         //歌词文件与 mp3 文件在同一目录，并且名字要相同
032         File f = new File(song path.replace(".mp3", ".lrc"));
033         try {
034             FileInputStream fis = new FileInputStream(f);
035             //以 utf-8 方式解析文字
036             InputStreamReader isr = new InputStreamReader(fis, "UTF-8");
037             BufferedReader br = new BufferedReader(isr);
038             String s = "";
039             while ((s = br.readLine()) != null) {
040                 //替换字符
```

```
041                 s = s.replace("[", "");
042                 s = s.replace("]", "@");
043                 //分离“@”字符
044                 String splitLrc_data[] = s.split("@");
045                 if (splitLrc_data.length > 1) {
046                     mLrcContent.setLrc(splitLrc_data[1]);
047                     //处理歌词取得歌曲时间
048                     int LrcTime = TimeStr(splitLrc_data[0]);
049                     //设置歌词时间
050                     mLrcContent.setLrc_time(LrcTime);
051                     //添加进列表数组
052                     LrcList.add(mLrcContent);
053                     //创建对象
054                     mLrcContent = new LrcContent();
055                 }
056             }
057             br.close();
058             isr.close();
059             fis.close();
060         } catch (FileNotFoundException e) {
061             // TODO Auto-generated catch block
062             e.printStackTrace();
063             stringBuilder.append("未找到歌词文件");
064         } catch (IOException e) {
065             // TODO Auto-generated catch block
066             e.printStackTrace();
067             stringBuilder.append("木有读取到歌词啊！");
068         }
069         return stringBuilder.toString();
070     }
071     /**
072      * 解析歌曲时间处理类
073      */
074     public int TimeStr(String timeStr) {
075         timeStr = timeStr.replace(":", ".");
076         timeStr = timeStr.replace(".", "@");
077         //取得时间信息存放到数组中
078         String timeData[] = timeStr.split("@");
079
080         //分离出分、秒并转换为整型
081         int minute = Integer.parseInt(timeData[0]);
082         int second = Integer.parseInt(timeData[1]);
083         int millisecond = Integer.parseInt(timeData[2]);
084
085         //计算上一行与下一行的时间并转换为毫秒数
086         int currentTime = (minute * 60 + second) * 1000 + millisecond *
            10;
087         return currentTime;
088     }
089     public List<LrcContent> getLrcContent() {
090         return LrcList;
091     }
092     /**
093      * 获得歌词和时间并返回的类
094      */
095     public class LrcContent {
096         private String Lrc;
097         private int Lrc_time;
098         //获得歌词
```

```
099         public String getLrc() {
100             return Lrc;
101         }
102         //设置歌词
103         public void setLrc(String lrc) {
104             Lrc = lrc;
105         }
106         //获得歌词对应的时间
107         public int getLrc time() {
108             return Lrc time;
109         }
110         //设置歌词对应的时间
111         public void setLrc_time(int lrc_time) {
112             Lrc_time = lrc_time;
113         }
114     }
115 }
```

17.4　主界面功能实现

17.4.1　音乐播放界面

整个音乐播放的过程基本都在主界面中完成，新建 MusicPlayerActivity.java，代码如下所示，刚开始声明一些需要用到的变量，如界面按钮、文本视图、图片视图等。接着在onCreate()函数中依次初始化界面元素，以供后续使用。代码 064 行通过调用 MusicListView()的函数 disPlayList 来为列表界面 musicListView 绑定适配器，与之类似的是代码 069 行通过调用 MusicSpecialView 中的 disPlaySpecial()函数来为专辑列表 musicGridView 设置适配器。

```
001 /**
002  * 音乐播放界面主类
003  * 强制竖屏：AndroidManifest 中设置 Activity 的 Screen Orientation
     为 portrait
004  */
005 public class MusicPlayerActivity extends Activity {
006     //3 个按钮：前一首、播放、后一首
007     ImageButton left_ImageButton;
008     public static ImageButton play_ImageButton;
009     ImageButton right_ImageButton;
010     //音乐列表
011     ListView musicListView;
012     //专辑列表
013     GridView musicGridView;
014     //上下文
015     public static Context context;
016     //初始化歌词检索值
017     public int Index = 0;
018     //为后台播放通知创建对象
019     public static NotificationManager mNotificationManager;
020     //绑定 SeekBar 和各种属性 TextView
021     public static SeekBar seekbar;
022     public static TextView time_left;
```

```
023     public static TextView time right;
024     public static TextView music Name;
025     public static LrcView lrc view;
026     public static TextView music Album;
027     public static TextView music Artist;
028     public static ImageView music AlbumArt;
029     public static TextView music number;
030     //左侧动画
031     public static RunGif runEql;
032     //为倒影创建对象
033     public RelativeLayout relativeflac;
034     public static ImageView reflaction;
035     //为歌曲时间和播放时间定义静态变量
036     public static int song_time = 0;
037     public static int play_time = 0;
038     //为类 Music_infoAdapter 声明静态变量
039     public static Music infoAdapter music info;
040     //声明两个页面对象
041     private BigDragableLuncher bigPage;
042     private DragableLuncher smallPage;
043     //声明按钮及对应的图片
044     private ImageButton[] blind btn = new ImageButton[3];
045     private int[] btn id = new int[] { R.id.imageButton1,
        R.id.imageButton2,
046             R.id.imageButton3 };
047     @Override
048     public void onCreate(Bundle savedInstanceState) {
049        super.onCreate(savedInstanceState);
050        setContentView(R.layout.main);
051        context = this;
052        bigPage = (BigDragableLuncher) findViewById(R.id.all space);
053        //中间滑动模块
054        smallPage = (DragableLuncher) findViewById(R.id.space);
055        //绑定 GIF 动画界面
056        runEql = (RunGif) findViewById(R.id.runGif1);
057        //倒影布局
058       relativeflac = (RelativeLayout) findViewById(R.id.relative
         layout1);  059reflaction = (ImageView) findViewById(R.id.
         inverted_view);
060        //创建对象获得系统服务
061        mNotificationManager = (NotificationManager) getSystemService
           (Context.NOTIFICATION_SERVICE);
062        // 绑定歌曲列表界面
063        musicListView = (ListView) findViewById(R.id.listView1);
064        new MusicListView().disPlayList(musicListView, this);
065        // 将获取的歌曲属性放到当前适配器中
066        music_info = new Music_infoAdapter(this);
067        // 绑定专辑列表界面
068        musicGridView = (GridView) findViewById(R.id.gridview1);
069        new MusicSpecialView().disPlaySpecial(musicGridView, this);
```

17.4.2　MusicListView 类

如下所示，是类 MusicListView 的代码，代码中通过传入 context 参数获得 Music_infoAdapter 实例，并将获得的适配器匹配到列表界面中。

```
01  package com.android.supermario;
```

```
02  import android.content.Context;
03  import android.widget.ListView;
04  //音乐播放列表
05  public class MusicListView {
06      ListView lv_1;
07      public static Music_infoAdapter m_info_1;
08      public void disPlayList(ListView lv, Context context) {
09          //取得音乐文件数据适配器
10          m_info_1 = new Music_infoAdapter(context);
11          this.lv_1 = lv;
12          //为音乐列表设置适配器
13          this.lv_1.setAdapter(m_info_1);
14      }
15  }
```

17.4.3　新建类 MusicSpecialView

与音乐播放列表类似，新建类 MusicSpecialView，在 disPlaySpecial()函数中通过传入的 context 参数获得 Music_infoAdapter 实例，并将获得的数据适配到界面中。如以下代码 34～70 行所示，在函数 getView()中将 Music_infoAdapter 中音乐列表的数据分别适配到膨胀出来的视图 gridspecial_item 中。

```
01  public class MusicSpecialView extends BaseAdapter {
02      public Context context;
03      GridView gv_1;
04      public static MusicSpecialView msv;
05  
06      public void disPlaySpecial(GridView gv, Context context) {
07          this.context = context;
08          //初始化视图
09          msv = new MusicSpecialView();
10          this.gv_1 = gv;
11          this.gv_1.setAdapter(msv);
12  
13      }
14      //获得音乐文件总数
15      @Override
16      public int getCount() {
17          // TODO Auto-generated method stub
18          return Music_infoAdapter.musicList.size();
19      }
20      //取得指定的文件
21      @Override
22      public Object getItem(int arg0) {
23          //TODO Auto-generated method stub
24          return Music_infoAdapter.musicList.get(arg0);
25      }
26      //取得指定的文件
27      @Override
28      public long getItemId(int arg0) {
29          //TODO Auto-generated method stub
30          return 0;
31      }
32      //获得每一个元素的视图
33      @Override
34      public View getView(int position, View convertView, ViewGroup parent)
```

```
        {
35          // TODO Auto-generated method stub
36          //膨胀出视图
37          LayoutInflater lif = LayoutInflater.from(Music
            PlayerActivity.context);
38          View v = lif.inflate(R.layout.gridspecial_item, null);
39          //每一行元素包含一个图片视图和两个文本视图
40          ImageView iv;
41          TextView tv1;
42          TextView tv2;
43          //初始化界面元素
44          iv = (ImageView) v.findViewById(R.id.gridspecial view1);
45          tv1 = (TextView) v.findViewById(R.id.album xxx);
46          tv2 = (TextView) v.findViewById(R.id.artist xxx);
47
48          //获取专辑图片路径
49          String url = MusicPlayerActivity.music_info
50                 .getAlbumArt(Music_infoAdapter.musicList.get
                    (position).get_id());
51          if (url != null) {
52              //设置专辑对应的图片
53              iv.setImageURI(Uri.parse(url));
54          } else {
55              //设置默认图片
56              iv.setImageResource(R.drawable.default_album);
57          }
58          iv.setAnimation(AnimationUtils.loadAnimation(
59                 MusicPlayerActivity.context, R.anim.alpha z));
60
61          //带动画显示专辑名
62          tv1.setText(Music infoAdapter.musicList.get(position)
            .getMusicAlbum());
63          tv1.setAnimation(AnimationUtils.loadAnimation(
64                 MusicPlayerActivity.context, R.anim.alpha y));
65
66          //带动画显示歌手名
67          tv2.setText(Music infoAdapter.musicList.get(position).
            getMusicSinger());
68          tv2.setAnimation(AnimationUtils.loadAnimation(
69                 MusicPlayerActivity.context, R.anim.alpha_y));
70          return v;
71      }
72  }
```

17.4.4　对界面初始化

继续回到音乐播放主界面初始化函数中，在初始化完左侧的音乐列表界面和右侧的音乐专辑界面之后，我们开始对当前界面进行初始化，如以下代码 104～190 行所示。首先为底部的 3 个切换按钮绑定监听器 ocl，监听器 ocl 实现如代码 053～103 行所示，根据传入的视图的资源 id 判断当前单击的是哪个按钮。当单击最左边一个按钮时，将屏幕显示为音乐播放列表界面，将当前按钮状态设置为“按下”，并根据按钮的当前状态为按钮重新设置背景图片，以区分出是否按下。中间按钮和右边按钮的情况与之类似。

设置完按键监听器之后，继续初始化未初始化的界面元素。代码 125 行开始对获取的

音乐数据进行解析，首先判断歌曲数量是否大于 0 以及是否都是以 mp3 结尾的文件，到此说明有可播放的音乐文件。接着将获得的音乐文件的相应信息如歌曲时间、歌曲名字、当前播放到第几首、歌手名字等显示到界面的指定位置中。

其中比较复杂的是显示专辑的图片，若歌曲信息中带有专辑图片的地址，则显示指定专辑图片，并为之创建倒影位图。若歌曲信息中不含有专辑图片信息，则使用默认图片，并为之创建倒影位图。

001～043 行为音乐播放列表和专辑列表绑定监听器，当单击指定音乐时，启动播放音乐服务播放指定的音乐。代码 194～216 行为音乐进度条设置改变监听器，当进度条发生改变时调用 onProgressChanged 函数将当前歌曲播放时间设置为指定的时间。代码 220～260 行为控制音乐播放的 3 个按钮，分别为“下一首”、“播放/暂停”、“上一首”，通过传入指定参数给音乐播放类，来达到控制音乐播放的目的。

```
001 //监听播放列表单击事件
002 musicListView.setOnItemClickListener(new OnItemClickListener() {
003     @Override
004     public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
005             long arg3) {
006         // TODO Auto-generated method stub
007         //打开播放音乐服务
008         Intent play_1 = new Intent(MusicPlayerActivity.this,
009                 ControlPlay.class);
010         //将控制参数传递给音乐播放服务
011         play 1.putExtra("control", "listClick");
012         play_1.putExtra("musicId_1", arg2);
013         startService(play_1);
014         //单击后动画跳转到播放界面
015         bigPage.setAnimation(AnimationUtils.loadAnimation(
016                 MusicPlayerActivity.this, R.anim.alpha_x));
017         bigPage.setToScreen(1);
018         blind btn[1]
019                 .setBackgroundResource(R.drawable.big button pressed);
020         blind_btn[0].setBackgroundResource(R.drawable.big_
        button style);
021     }
022 });
023 //监听专辑列表单击事件
024 musicGridView.setOnItemClickListener(new OnItemClickListener() {
025     @Override
026     public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
027             long arg3) {
028         // TODO Auto-generated method stub
029         //打开音乐播放服务
030         Intent play 2 = new Intent(MusicPlayerActivity.this,
031                 ControlPlay.class);
032         //传递专辑信息和控制信息
033         play_2.putExtra("control", "gridClick");
034         play 2.putExtra("musicId 2", arg2);
035         startService(play 2);
036         //单击后动画跳转到播放界面
037         bigPage.setAnimation(AnimationUtils.loadAnimation(
038                 MusicPlayerActivity.this, R.anim.alpha x));
039         bigPage.setToScreen(1);
```

```
040         blind btn[1].setBackgroundResource(R.drawable.big button
           pressed);
041         blind_btn[2].setBackgroundResource(R.drawable.big_
           button_style);
042     }
043 });
044 //小页面允许触摸执行动画
045 smallPage.isOpenTouchAnima(true);
046 //绑定 ImageButton
047 for (int k = 0; k < blind btn.length; k++) {
048     blind btn[k] = (ImageButton) findViewById(btn id[k]);
049 }
050 //设置按钮被选中颜色和默认颜色
051 bigPage.setBottomBarBg(blind btn, Color.GREEN, Color.YELLOW);
052 //判断单击了哪个按钮并执行跳转界面
053 android.view.View.OnClickListener ocl = new View.OnClickListener()
    {
054     @Override
055     public void onClick(View v) {
056         // TODO Auto-generated method stub
057         switch (v.getId()) {
058         //音乐列表
059         case R.id.imageButton1:
060             bigPage.setAnimation(AnimationUtils.loadAnimation(
061                     MusicPlayerActivity.this, R.anim.alpha_x));
062             bigPage.setToScreen(0);
063             //设置当前按钮的状态为按下
064             v.setPressed(true);
065             //设置 3 个按钮的背景图片
066             blind btn[0]
067                     .setBackgroundResource(R.drawable.big button
                     pressed);
068             blind btn[1]
069                    .setBackgroundResource(R.drawable.big_button_style);
070             blind_btn[2]
071                    .setBackgroundResource(R.drawable.big button style);
072             break;
073         //正在播放
074         case R.id.imageButton2:
075             bigPage.setAnimation(AnimationUtils.loadAnimation(
076                     MusicPlayerActivity.this, R.anim.alpha_x));
077             bigPage.setToScreen(1);
078             //设置当前按钮的状态为按下
079             v.setPressed(true);
080             //设置 3 个按钮的背景图片
081             blind btn[1]
082                     .setBackgroundResource(R.drawable.big button
                     pressed);
083             blind btn[0]
084                     .setBackgroundResource(R.drawable.big_button_
                     style);
085             blind_btn[2]
086                    .setBackgroundResource(R.drawable.big button style);
087             break;
088         case R.id.imageButton3:
089             bigPage.setAnimation(AnimationUtils.loadAnimation(
090                     MusicPlayerActivity.this, R.anim.alpha x));
091             bigPage.setToScreen(2);
```

```
092             //设置当前按钮的状态为按下
093             v.setPressed(true);
094             //设置 3 个按钮的背景图片
095             blind btn[2]
096                    .setBackgroundResource(R.drawable.big_button_
                       pressed);
097             blind btn[1]
098                    .setBackgroundResource(R.drawable.big button
                        style);
099             blind btn[0]
100                    .setBackgroundResource(R.drawable.big button
                        style);
101          }
102       }
103 };
104 //初始化设置
105 blind_btn[1].setBackgroundResource(R.drawable.big_button_pressed);
106 //3 个按钮分别指定监听器
107 blind_btn[0].setOnClickListener(ocl);
108 blind_btn[1].setOnClickListener(ocl);
109 blind_btn[2].setOnClickListener(ocl);
110 //初始化按钮
111 left_ImageButton = (ImageButton) findViewById(R.id.ib1);
112 play_ImageButton = (ImageButton) findViewById(R.id.ib2);
113 right_ImageButton = (ImageButton) findViewById(R.id.ib3);
114 //初始化界面其他元素
115 time left = (TextView) findViewById(R.id.time tv1);
116 time right = (TextView) findViewById(R.id.time tv2);
117 music Name = (TextView) findViewById(R.id.music name);
118 music Album = (TextView) findViewById(R.id.music album);
119 music Artist = (TextView) findViewById(R.id.music artist);
120 seekbar = (SeekBar) findViewById(R.id.player seekbar);
121 lrc view = (LrcView) findViewById(R.id.LyricShow);
122 music AlbumArt = (ImageView) findViewById(R.id.music AlbumArt);
123 music_number = (TextView) findViewById(R.id.music_number);
124
125 //判断歌曲不能为空并且后缀为.mp3
126 if (music_info.getCount() > 0
127       && Music_infoAdapter.musicList.get(ControlPlay.playing_id)
128              .getMusicName().endsWith(".mp3")) {
129    //显示获取的歌曲时间
130    time_right.setText(Music_infoAdapter
131          .toTime(Music_infoAdapter.musicList.get(
132                ControlPlay.playing_id).getMusicTime()));
133    //截取.mp3 字符串
134    String a = Music infoAdapter.musicList.get(ControlPlay.playing id)
135          .getMusicName();
136    int b = a.indexOf(".mp3");
137    String c = a.substring(0, b);
138    //显示获取的歌曲名
139    music_Name.setText(c);
140    music_Name.setAnimation(AnimationUtils.loadAnimation(
141          MusicPlayerActivity.this, R.anim.translate_z));
142
143    //显示播放当前第几首和歌曲总数
144    int x = ControlPlay.playing_id + 1;
145    music_number.setText("" + x + "/"
146          + Music_infoAdapter.musicList.size());
147
```

```
148     //显示获取的艺术家
149     music_Artist.setText(Music_infoAdapter.musicList.get(
150             ControlPlay.playing_id).getMusicSinger());
151     //获取专辑图片路径
152     String url = MusicPlayerActivity.music_info
153             .getAlbumArt(Music_infoAdapter.musicList.get(
154                     ControlPlay.playing_id).get_id());
155     if (url != null) {
156         //显示获取的专辑图片
157         music_AlbumArt.setImageURI(Uri.parse(url));
158         music_AlbumArt.setAnimation(AnimationUtils.loadAnimation(
159                 context, R.anim.alpha_z));
160         try {
161             /* 为倒影创建位图 */
162             Bitmap mBitmap = BitmapFactory.decodeFile(url);
163             reflaction.setImageBitmap(createReflectedImage(mBitmap));
164             reflaction.setAnimation(AnimationUtils.loadAnimation(
165                     context, R.anim.alpha_z));
166         } catch (Exception e) {
167             // TODO Auto-generated catch block
168             e.printStackTrace();
169         }
170     } else {
171         //设置显示为默认图片
172         music_AlbumArt.setImageResource(R.drawable.album);
173         music_AlbumArt.setAnimation(AnimationUtils.loadAnimation(
174                 context, R.anim.alpha_z));
175         try {
176             /* 为倒影创建位图 */
177             Bitmap mBitmap = ((BitmapDrawable) getResources()
178                     .getDrawable(R.drawable.album)).getBitmap();
179             reflaction.setImageBitmap(createReflectedImage(mBitmap));
180             reflaction.setAnimation(AnimationUtils.loadAnimation(
181                     context, R.anim.alpha_z));
182         } catch (Exception e) {
183             // TODO Auto-generated catch block
184             e.printStackTrace();
185         }
186     }
187 } else {
188     Toast.makeText(MusicPlayerActivity.this, "手机里木有找到歌曲哦！",
189             Toast.LENGTH_LONG).show();
190 }
191 /**
192  * 监听拖动 SeekBar 事件
193  */
194 seekbar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {
195     @Override
196     public void onStopTrackingTouch(SeekBar seekBar) {
197         // TODO Auto-generated method stub
198 
199     }
200     @Override
201     public void onStartTrackingTouch(SeekBar seekBar) {
202         // TODO Auto-generated method stub
203 
204     }
205 
206     @Override
207     public void onProgressChanged(SeekBar seekBar, int progress,
```

```
208             boolean fromUser) {
209         // TODO Auto-generated method stub
210         // 判断用户是否触拖 SeekBar 并且不为空才执行
211         if (fromUser && ControlPlay.myMediaPlayer != null) {
212             ControlPlay.myMediaPlayer.seekTo(progress);
213         }
214         time_left.setText(Music_infoAdapter.toTime(progress));
215     }
216 });
217 /**
218  * 监听“上一首”并实现功能
219  */
220 left ImageButton.setOnClickListener(new ImageButton.
    OnClickListener() {
221
222     @Override
223     public void onClick(View v) {
224         // TODO Auto-generated method stub
225         //打开音乐播放服务
226         Intent play left = new Intent(MusicPlayerActivity.this,
227                 ControlPlay.class);
228         play_left.putExtra("control", "front");
229         startService(play left);
230     }
231 });
232 /**
233  * 监听“播放”并实现功能
234  */
235 play ImageButton.setOnClickListener(new ImageButton.OnClickListener()
    {
236     @Override
237     public void onClick(View v) {
238         // TODO Auto-generated method stub
239         //打开音乐播放服务
240         Intent play center = new Intent(MusicPlayerActivity.this,
241                 ControlPlay.class);
242         play_center.putExtra("control", "play");
243         startService(play_center);
244     }
245
246 });
247 /**
248  * 监听“下一首”并实现功能
249  */
250 right ImageButton.setOnClickListener(new ImageButton.OnClickListener
    () {
251     @Override
252     public void onClick(View v) {
253         // TODO Auto-generated method stub
254         //打开音乐播放服务
255         Intent play right = new Intent(MusicPlayerActivity.this,
256                 ControlPlay.class);
257         play_right.putExtra("control", "next");
258         startService(play right);
259     }
260 });
261
262}
```

17.4.5　图片的设置

在上述初始化函数中用到了几个自定义的函数，代码如下所示。首先是createReflectedImage()函数用于为图片创建倒影，首先获得一个宽度与原始图片一致，高度为原始图片一半的图片，该图片与原始图片沿 Y 轴成镜像。接着创建一个高度为原始图片高度 3/2 的画板，在上面从上到下依次画上原始图片、4dp 的间隔矩形、镜像图片，接着通过 LinearGradient 产生渐变效果，并应用于生成的镜像图片，这就产生了最终的镜像效果。

当用户按下 BACK 键时，将打开一个对话框，提示用户是退出还是返回，如果用户选择退出，则关闭音乐播放服务和取消状态栏的通知，并将此程序退出。若用户单击返回或者对话框之外的区域，将关闭对话框。

```
001     /**
002      * 倒影的实现方法
003      *
004      * @param originalBitmap
005      * @return
006      */
007     static Bitmap createReflectedImage(Bitmap originalBitmap) {
008         // TODO Auto-generated method stub
009 
010         //图片与倒影的间隔距离
011         final int reflectionGap = 4;
012         //图片的宽度
013         int width = originalBitmap.getWidth();
014         //图片的高度
015         int height = originalBitmap.getHeight();
016 
017         Matrix matrix = new Matrix();
018         //图片缩放，x 轴变为原来的 1 倍，y 轴为-1 倍,实现图片的反转
019         matrix.preScale(1, -1);
020         //创建反转后的图片 Bitmap 对象，图片高度是原图的一半
021         Bitmap reflectionBitmap = Bitmap.createBitmap(originalBitmap,0,
022                 height / 2, width, height / 2, matrix, false);
023         //创建标准的 Bitmap 对象，宽度和原图一致，高度是原图的 1.5 倍
024         Bitmap withReflectionBitmap = Bitmap.createBitmap(width,
        (height
025                 + height / 2 + reflectionGap), Config.ARGB_8888);
026 
027         //构造函数传入 Bitmap 对象，为了在图片上画图
028         Canvas canvas = new Canvas(withReflectionBitmap);
029         //画原始图片
030         canvas.drawBitmap(originalBitmap, 0, 0, null);
031 
032         //画间隔矩形
033         Paint defaultPaint = new Paint();
034         canvas.drawRect(0, height, width, height + reflectionGap,
        defaultPaint);
035 
036         //画倒影图片
037         canvas.drawBitmap(reflectionBitmap, 0, height + reflection
```

```
            Gap,null);
038
039         //实现倒影效果
040         Paint paint = new Paint();
041         LinearGradient shader = new LinearGradient(0,
042                 originalBitmap.getHeight(), 0,
043                 withReflectionBitmap.getHeight(), 0x70ffffff,
                0x00ffffff,
044                 TileMode.MIRROR);
045         paint.setShader(shader);
046         paint.setXfermode(new PorterDuffXfermode(Mode.DST_IN));
047
048         //覆盖效果
049         canvas.drawRect(0, height, width, withReflection
            Bitmap.getHeight(),
050                 paint);
051
052         return withReflectionBitmap;
053     }
054
055     /**
056      * 按下返回键后，提示用户是否退出程序
057      */
058     @Override
059     public boolean onKeyDown(int keyCode, KeyEvent event) {
060         //TODO Auto-generated method stub
061
062         if (keyCode == KeyEvent.KEYCODE_BACK) {
063
064             Dialog dialog = new MyDialog(MusicPlayerActivity.this,
065                     R.style.MyDialog);
066             //设置触摸对话框以外的地方取消对话框
067             dialog.setCanceledOnTouchOutside(true);
068             dialog.show();
069         }
070         return super.onKeyDown(keyCode, event);
071     }
072
073     /**
074      * 后台提示播放通知的方法
075      * 需要在AndroidManifest的C_MusicPlayerActivity中添加android:
        launchMode=
076      * "singleTop"才可以完全退出
077      *
078      * @param tickerText
079      *            传入的歌曲名
080      * @param title
081      * @param content
082      * @param drawable
083      *            图片路径
084      */
085     public static void setNotice(String tickerText, String title,
086             String content, int drawable) {
087         //创建一个通知对象，传入相应的参数
088         Notification notification = new Notification(drawable
            tickerText,
089                 System.currentTimeMillis());
090         //设置通知不能被清除
```

```
091         notification.flags = Notification.FLAG NO CLEAR;
092         //封装的 Intent 跳转，由系统来决定什么时候启动跳转 Intent
093         PendingIntent contentIntent = PendingIntent.getActivity(context
            , 0,
094                 new Intent(context, MusicPlayerActivity.class), 0);
095         //notification.setLatestEventInfo(this, "通知标题","通知内容",
            intent);
096         notification.setLatestEventInfo(context, title, content,
            contentIntent);
097         //发送通知，代码应该放到最后，否则会报错，显示的 Notification 都有一个唯
            一的 ID 是 1
098         mNotificationManager.notify(1, notification);
099     }
100
101     /**
102      * 自定义对话框的类
103      */
104     class MyDialog extends Dialog {
105         Context context;
106         //构造函数
107         public MyDialog(Context context) {
108             super(context);
109             // TODO Auto-generated constructor stub
110             this.context = context;
111         }
112         //构造函数
113         public MyDialog(Context context, int theme) {
114             super(context, theme);
115             // TODO Auto-generated constructor stub
116             this.context = context;
117         }
118
119         @Override
120         protected void onCreate(Bundle savedInstanceState) {
121             // TODO Auto-generated method stub
122             super.onCreate(savedInstanceState);
123             this.setContentView(R.layout.dialog);
124             //“退出”按钮和“返回”按钮
125             Button exit_button, return_button;
126             //初始化
127             exit button = (Button) findViewById(R.id.exit button2);
128             return button = (Button) findViewById(R.id.return button3);
129
130             //结束服务退出应用程序
131             exit_button.setOnClickListener(new View.OnClickListener() {
132
133                 @Override
134                 public void onClick(View v) {
135                     // TODO Auto-generated method stub
136                     //关掉音乐播放服务
137                     stopService(new Intent(MusicPlayerActivity.this,
138                             ControlPlay.class));
139                     mNotificationManager.cancel(1);
140                     //退出程序
141                     System.exit(0);
142                 }
143             });
144
145             //返回
```

```
146             eturn button.setOnClickListener(new View.OnClickListener() {
147                 @Override
148                 public void onClick(View v) {
149                     // TODO Auto-generated method stub
150                     //关闭对话框
151                     dismiss();
152                 }
153             });
154         }
155     }
```

17.4.6　布局文件 dialog.xml

对话框界面如图 17.5 所示，新建对话框布局文件 dialog.xml 代码如下所示。最上面显示一个标题，标题旁边是用 ImageView 显示的一个 Icon。下面依次显示消息内容、分割线和两个按钮。



图 17.5　对话框界面

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout
03    xmlns:android="http://schemas.android.com/apk/res/android"
04    android:orientation="vertical"
05    android:layout_width="wrap_content"
06    android:layout_height="wrap_content"
07    android:background="#00000000"
08    android:gravity="center_vertical|center_horizontal">
09    <RelativeLayout
```

```
10          android:layout width="280dip"
11          android:layout height="152dip"
12          android:background="#eeffffff" >
13         <!-- 对话框顶部背景 -->
14          <ImageView
15              android:id="@+id/dlg_top"
16              android:layout_width="fill_parent"
17              android:layout_height="40dip"
18              android:background="@drawable/dlg_top_background" />
19        <!-- 对话框标题旁边 icon -->
20          <ImageView
21              android:id="@+id/imageView1"
22              android:layout width="37.6dip"
23              android:layout height="37.6dip"
24              android:layout marginLeft="10dip"
25              android:layout marginTop="2dip"
26              android:src="@drawable/dlg untitled" />
27          <!-- 对话框标题 -->
28          <TextView
29              android:layout width="wrap content"
30              android:layout_height="wrap_content"
31              android:layout marginLeft="60dip"
32              android:layout marginTop="7dip"
33              android:text="MP3 播放器"
34              android:textColor="#ffffff"
35              android:textSize="20sp" />
36          <!-- 对话框消息 -->
37          <TextView
38              android:id="@+id/tv1"
39              android:layout width="wrap content"
40              android:layout height="wrap content"
41              android:layout below="@id/dlg top"
42              android:layout marginLeft="30dip"
43              android:layout marginTop="10dip"
44              android:text="确定要退出? "
45              android:textColor="#333399"
46              android:textSize="22dp"
47              android:textStyle="bold" />
48          <!-- 分隔线 -->
49          <View
50              android:layout width="fill parent"
51              android:layout height="1px"
52              android:layout below="@id/tv1"
53              android:layout marginTop="12dip"
54              android:background="#6f999999" />
55          <!—“退出”按钮 -->
56          <Button
57              android:id="@+id/exit_button2"
58              android:layout_width="70dip"
59              android:layout height="32dip"
60              android:layout alignLeft="@+id/imageView1"
61              android:layout below="@+id/tv1"
62              android:layout marginTop="20dp"
63              android:background="@drawable/dlg button style"
64              android:text="退出"
65              android:textColor="#b1ffffff"
66              android:textSize="18sp" />
67          <!—“返回”按钮 -->
68          <Button
69              android:id="@+id/return button3"
```

```
70            android:layout_width="70dip"
71            android:layout_height="32dip"
72            android:layout_alignTop="@+id/exit_button2"
73            android:layout_toRightOf="@+id/tv1"
74            android:background="@drawable/dlg_button_style"
75            android:text="返回"
76            android:textColor="#b1ffffff"
77            android:textSize="18sp" />
78      </RelativeLayout>
79    </LinearLayout>
```

17.5　歌曲信息类

在界面视图数据方面最核心的一个类就是歌曲信息类 Music_infoAdapter，通过这个类我们可以知道当前曲库中的歌曲信息，并为界面的其他视图提供数据。如下所示为该类的实现代码，如代码 007 行所示，该类中声明了静态变量 musicList 用于存放音乐信息，在构造函数中通过查询本地媒体库，获得本地所有音乐的信息并存储到 musicList 中。这些信息包括歌曲名、歌手名字、歌曲时间、专辑名、歌曲路径、歌曲 ID 等信息。

函数 getAlbumArt()用于根据歌曲的 id 获得专辑图片地址，具体实现办法是，先通过歌曲 id 查询到专辑的 id，再通过专辑的 id 获得专辑图片的地址并返回。主界面正是通过调用该方法来获得专辑图片并显示到滑动视图中间。

```
001 public class Music_infoAdapter extends BaseAdapter {
002     //用来获得 ContentProvider（共享数据库）
003     public ContentResolver cr;
004     //用来装载查询到的音乐文件数据
005     public Cursor mCursor;
006     //歌曲列表信息
007     public static List<MusicInfomation> musicList;
008     public MusicInfomation mi;
009
010     public Context context;
011
012     //音乐信息：1、歌曲名，2、歌手，3、歌曲时间，4、专辑（专辑图片、专辑名称、
          专辑 ID[用来获取图片]），5、歌曲大小
013     public Music_infoAdapter(Context context) {
014         this.context = context;
015         // 取得数据库对象
016         cr = context.getContentResolver();
017         //初始化音乐列表数组
018         musicList = new ArrayList<MusicInfomation>();
019         String[] s1 = new String[] {
020                 //歌曲名
021                 MediaStore.Audio.Media.DISPLAY_NAME,
022                 //专辑名
023                 MediaStore.Audio.Media.ALBUM,
024                 //歌手名
025                 MediaStore.Audio.Media.ARTIST,
026                 //歌曲时间
```

```
027                 MediaStore.Audio.Media.DURATION,
028                 //歌曲大小
029                 MediaStore.Audio.Media.SIZE,
030                 //歌曲 ID
031                 MediaStore.Audio.Media._ID,
032                 //歌曲路径
033                 MediaStore.Audio.Media.DATA };
034
035         //查询所有音乐信息
036         mCursor = cr.query(MediaStore.Audio.Media.EXTERNAL_CONTENT_
        URI, s1,null, null, null);
037
038
039         if (mCursor != null) {
040             //移动到第一个
041             mCursor.moveToFirst();
042             //获得歌曲的各种属性
043             for (int i = 0; i < mCursor.getCount(); i++) {
044                 //过滤 mp3 文件
045                 if (mCursor.getString(0).endsWith(".mp3")) {
046
047                     mi = new MusicInfomation();
048                     mi.setMusicName(mCursor.getString(0));
049                     mi.setMusicAlbum(mCursor.getString(1));
050                     mi.setMusicSinger(mCursor.getString(2));
051                     mi.setMusicTime(mCursor.getInt(3));
052                     mi.setMusicSize(mCursor.getInt(4));
053                     mi.set_id(mCursor.getInt(5));
054                     mi.setMusicPath(mCursor.getString(6));
055                     //装载到列表中
056                     musicList.add(mi);
057                 }
058                 //移动到下一个
059                 mCursor.moveToNext();
060             }
061         }
062     }
063     //获得数量
064     @Override
065     public int getCount() {
066         // TODO Auto-generated method stub
067         return musicList.size();
068     }
069     //获得指定元素
070     @Override
071     public Object getItem(int arg0) {
072         // TODO Auto-generated method stub
073         return musicList.get(arg0);
074     }
075     //获得指定元素
076     @Override
077     public long getItemId(int arg0) {
078         // TODO Auto-generated method stub
079         return 0;
080     }
081     //获得视图
082     @Override
083     public View getView(int arg0, View arg1, ViewGroup arg2) {
084         // TODO Auto-generated method stub
```

```
085         //膨胀出视图
086         LayoutInflater li = LayoutInflater.from(context);
087         View v = li.inflate(R.layout.musiclist item, null);
088
089         TextView tv1;
090         TextView tv2;
091         TextView tv3;
092         TextView tv4;
093         //初始化界面元素
094         tv1 = (TextView) v.findViewById(R.id.tv1);
095         tv2 = (TextView) v.findViewById(R.id.tv2);
096         tv3 = (TextView) v.findViewById(R.id.tv3);
097         tv4 = (TextView) v.findViewById(R.id.tv4);
098         //为界面元素设置内容
099         tv1.setText(musicList.get(arg0).getMusicName());
100         tv2.setText(toTime(musicList.get(arg0).getMusicTime()));
101         tv3.setText(musicList.get(arg0).getMusicPath());
102         tv4.setText(toMB(musicList.get(arg0).getMusicSize()) + "MB");
103         return v;
104     }
105
106     /**
107      * 时间转化处理
108      */
109     public static String toTime(int time) {
110         time /= 1000;
111         int minute = time / 60;
112         int second = time % 60;
113         minute %= 60;
114         //格式化时间
115         return String.format(" %02d:%02d ", minute, second);
116     }
117
118     /**
119      * 文件大小转换，将 B 转换为 MB
120      */
121     public String toMB(int size) {
122         float a = (float) size / (float) (1024 * 1024);
123         String b = Float.toString(a);
124         int c = b.indexOf(".");
125         String fileSize = "";
126         fileSize += b.substring(0, c + 2);
127         return fileSize;
128     }
129
130     /**
131      * 歌曲专辑图片显示，如果有歌曲图片，才会返回，否则为 null，要注意判断
132      *
133      * @param trackId 是音乐的 id
134      * @return 返回类型是 String 类型的图片地址，也就是 uri
135      */
136     public String getAlbumArt(int trackId) {
137         //根据音乐的 id 获得专辑图片的 id
138         String mUriTrack = "content://media/external/audio/media/#";
139         String[] projection = new String[] { "album id" };
140         String selection = " id = ?";
141         String[] selectionArgs = new String[] { Integer.toString
            (trackId) };
142         Cursor mcCursor = context.getContentResolver().query(
```

```
143         Uri.parse(mUriTrack), projection,selection,selectionArgs,
144                null);
145         int album_id = 0;
146         if (mcCursor.getCount() > 0 && mcCursor.getColumnCount() > 0) {
147             mcCursor.moveToNext();
148             album_id = mcCursor.getInt(0);
149         }
150         mcCursor.close();
151         mcCursor = null;
152
153         if (album_id < 0) {
154             return null;
155         }
156         //根据专辑图片的 id 获得专辑图片的 uri 地址
157         String mUriAlbums = "content://media/external/audio/albums";
158         projection = new String[] { "album_art" };
159         mcCursor = context.getContentResolver().query(
160                Uri.parse(mUriAlbums + "/" + Integer.toString(album_id)),
161                projection, null, null, null);
162         String album_art = null;
163         if (mcCursor.getCount() > 0 && mcCursor.getColumnCount() > 0) {
164             mcCursor.moveToNext();
165             album_art = mcCursor.getString(0);
166         }
167         mcCursor.close();
168         mcCursor = null;
169
170         return album_art;
171     }
```

该类中还包含了一个 MusicInfomation 用于存放音乐的信息，代码如下所示，存放的信息包括歌曲名、歌手名字、歌曲时间、专辑 id、音乐路径等信息。

```
01/**
02  * 1、歌曲名 2、歌手 3、歌曲时间 4、专辑（专辑图片、专辑名称、专辑 ID[用
    来获取图片]） 5、歌曲大小
03  */
04 public class MusicInfomation {
05
06     private int  _id;
07     private String musicName;
08     private String musicSinger;
09     private int musicTime;
10     private String musicAlbum;
11     private int musicSize;
12     private String musicPath;
13     //取得 id
14     public int get_id() {
15         return _id;
16     }
17     //设置 id
18     public void set_id(int  _id) {
19         this._id =  _id;
20     }
21     //取得歌曲名
22     public String getMusicName() {
23         return musicName;
24     }
25     //设置歌曲名
```

```
26     public void setMusicName(String musicName) {
27         this.musicName = musicName;
28     }
29     //取得歌手名
30     public String getMusicSinger() {
31         return musicSinger;
32     }
33     //设置歌手名
34     public void setMusicSinger(String musicSinger) {
35         this.musicSinger = musicSinger;
36     }
37     //取得音乐播放总时间
38     public int getMusicTime() {
39         return musicTime;
40     }
41     //设置音乐播放总时间
42     public void setMusicTime(int musicTime) {
43         this.musicTime = musicTime;
44     }
45     //取得音乐相册
46     public String getMusicAlbum() {
47         return musicAlbum;
48     }
49     //设置音乐相册
50     public void setMusicAlbum(String musicAlbum) {
51         this.musicAlbum = musicAlbum;
52     }
53     //取得音乐大小
54     public int getMusicSize() {
55         return musicSize;
56     }
57     //设置音乐大小
58     public void setMusicSize(int musicSize) {
59         this.musicSize = musicSize;
60     }
61     //取得音乐路径
62     public String getMusicPath() {
63         return musicPath;
64     }
65     //设置音乐路径
66     public void setMusicPath(String musicPath) {
67         this.musicPath = musicPath;
68     }
69 }
```

17.6　音乐播放服务

新建 ControlPlay.java 代码如下所示，在类的开始声明了成员变量 mMediaPlayer 用于控制音乐播放，成员变量 c_ma 用于控制主界面视图，mLrcProcess 和 mLrcView 用于控制歌词的显示。

在服务初始化函数中调用 initMediaSource 默认播放第一首歌曲，接着在 onStart()函数中根据传递过来的控制参数调用相应的函数，play 表示直接播放当前歌曲，next 将调用 playNext()函数播放下一曲，front 将调用 playFront()函数播放前一首，如果传递的参数是

listClick 或者是 gridClick，则会根据传递的音乐的 id 播放对应的音乐。

```
001 //音乐播放服务
002 public class ControlPlay extends Service {
003     //媒体播放类
004     public static MediaPlayer myMediaPlayer;
005     public MusicPlayerActivity c_ma = new MusicPlayerActivity();
006     //歌词控制类
007     public LrcProcess mLrcProcess;
008     public LrcView mLrcView;
009     public static int playing id = 0;
010 
011     //初始化歌词检索值
012     private int index = 0;
013     //初始化歌曲播放时间的变量
014     private int CurrentTime = 0;
015     //初始化歌曲总时间的变量
016     private int CountTime = 0;
017     //创建对象
018     private List<LrcContent> lrcList = new ArrayList<LrcContent>();
019     //歌曲信息
020     public static Music infoAdapter m in;
021     Handler handler = new Handler();
022     public boolean playFlag = true;
023 
024     @Override
025     public IBinder onBind(Intent intent) {
026         // TODO Auto-generated method stub
027         return null;
028     }
029     //服务创建时调用
030     @Override
031     public void onCreate() {
032         // TODO Auto-generated method stub
033         super.onCreate();
034         initMediaSource(initMusicUri(0));
035     }
036     //服务销毁时调用
037     @Override
038     public void onDestroy() {
039         // TODO Auto-generated method stub
040 
041         super.onDestroy();
042         if (myMediaPlayer != null) {
043             //释放资源
044             myMediaPlayer.stop();
045             myMediaPlayer.release();
046             myMediaPlayer = null;
047         }
048     }
049     //服务启动时调用
050     @Override
051     public void onStart(Intent intent, int startId) {
052         // TODO Auto-generated method stub
053         super.onStart(intent, startId);
054         //获得控制标志字
055         String playFlag = intent.getExtras().getString("control");
056         //播放/暂停
057         if ("play".equals(playFlag)) {
```

```
058             playMusic();
059         } else if ("next".equals(playFlag)) {
060             //播放下一首
061             playNext();
062         } else if ("front".equals(playFlag)) {
063             //播放前一首
064             playFront();
065         } else if ("listClick".equals(playFlag)) {
066             //播放指定音乐
067             playing_id = intent.getExtras().getInt("musicId_1");
068             initMediaSource(initMusicUri(playing_id));
069             playMusic();
070         } else if ("gridClick".equals(playFlag)) {
071             //播放指定音乐
072             playing_id = intent.getExtras().getInt("musicId_2");
073             initMediaSource(initMusicUri(playing_id));
074             playMusic();
075         }
076     }
077     /**
078      * 初始化媒体对象
079      *
080      * @param mp3Path
081      *           mp3 路径
082      */
083     public void initMediaSource(String mp3Path) {
084         //解析歌曲 uri 地址
085         Uri mp3Uri = Uri.parse(mp3Path);
086         if (myMediaPlayer != null) {
087             myMediaPlayer.stop();
088             myMediaPlayer.reset();
089             myMediaPlayer = null;
090         }
091         //为 myMediaPlayer 创建对象
092         myMediaPlayer = MediaPlayer.create(this, mp3Uri);
093     }
094 
095     /**
096      * 返回列表第几行的歌曲路径
097      *
098      * @param _id
099      *           表示歌曲序号，从 0 开始
100      */
101     public String initMusicUri(int  _id) {
102         playing_id = _id;
103         String s;
104         //判断列表和列表长度不为空时才获取
105         if (Music_infoAdapter.musicList != null
106                 && Music_infoAdapter.musicList.size() != 0) {
107             s = Music_infoAdapter.musicList.get(_id).getMusicPath();
108             return s;
109         } else {
110             //否则返回空字符串
111             return "";
112         }
113     }
```

播放音乐的核心控制函数是 playMusic()，代码如下所示。首先判断播放器是否正在播

放音乐，如果正在播放则调用 pause 将歌曲暂停，同时停止动画的更新和通知的显示，并改变“播放”按钮的图片；否则继续播放音乐，打开动画更新标志。

当开始播放音乐时，首先要初始化歌词配置，载入歌词文件并启动线程显示歌词。同时考虑到播放过程中会更换歌曲，因此每次调用该函数时需要重新读取歌曲信息，如歌手名字、专辑名字等，并更新到指定视图中，更新方法与前面介绍的主界面初始化界面方法类似。

此外，还要设置歌曲播放完毕监听器，当一首歌播放完毕之后要调用 playNext()函数播放下一首歌曲。playNext()函数首先判断当前是否是最后一首歌，如果是则提示“已经是最后一首”，否则初始化下一首歌曲并调用 playMusic 播放，playFront()函数与之类似。

```
001 /**
002  * 音乐播放方法，并且带有暂停方法
003  */
004 public void playMusic() {
005
006     //判断歌曲不能为空
007
008     if (myMediaPlayer != null && Music_infoAdapter.musicList.size()
       != 0) {
009         if (myMediaPlayer.isPlaying()) {
010             //歌曲暂停
011             myMediaPlayer.pause();
012             //暂停更新 GIF 动画
013             MusicPlayerActivity.runEql.setFlag(false);
014             MusicPlayerActivity.runEql.invalidate();
015             //更换“播放”按钮背景
016             MusicPlayerActivity.play_ImageButton
017                     .setBackgroundResource(R.drawable.play_button);
018             //取消通知
019             MusicPlayerActivity.mNotificationManager.cancel(1);
020         } else {
021             myMediaPlayer.start();
022
023             //初始化歌词配置
024             mLrcProcess = new LrcProcess();
025             //读取歌词文件
026             mLrcProcess.readLRC(Music_infoAdapter.musicList.
            get(playing_id)
027                     .getMusicPath());
028             //传回处理后的歌词文件
029             lrcList = mLrcProcess.getLrcContent();
030             MusicPlayerActivity.lrc_view.setSentenceEntities(lrcList);
031             //切换带动画显示歌词
032             MusicPlayerActivity.lrc_view.setAnimation(AnimationUtils
033                     .loadAnimation(ControlPlay.this, R.anim.alpha_z));
034            //启动线程
035             mHandler.post(mRunnable);
036
037             //更换背景
038             MusicPlayerActivity.play_ImageButton
039                     .setBackgroundResource(R.drawable.pause_button);
040             //启动线程更新 SeekBar
041             startSeekBarUpdate();
042             //启动更新 GIF 动画
```

```
043             MusicPlayerActivity.runEql.setFlag(true);
044             MusicPlayerActivity.runEql.invalidate();
045
046             //更新歌曲播放第几首
047             int x = playing_id + 1;
048             MusicPlayerActivity.music_number.setText("" + x + "/"
049                     + Music_infoAdapter.musicList.size());
050
051             //截取.mp3 字符串
052             String a = Music infoAdapter.musicList.get(playing id)
053                     .getMusicName();
054             int b = a.indexOf(".mp3");
055             String c = a.substring(0, b);
056             //切换带动画更新歌曲名
057             MusicPlayerActivity.music_Name.setText(c);
058             MusicPlayerActivity.music_Name.setAnimation(AnimationUtils
059                     .loadAnimation(ControlPlay.this, R.anim.
                        translate_z));
060
061             //带动画更新专辑名
062             MusicPlayerActivity.music Album
063                   .setText(Music infoAdapter.musicList.get(playing id)
064                           .getMusicAlbum());
065             MusicPlayerActivity.music Album.setAnimation
                (AnimationUtils
066                     .loadAnimation(ControlPlay.this, R.anim.alpha y));
067
068             //更新歌手名
069             MusicPlayerActivity.music_Artist
070                   .setText(Music infoAdapter.musicList.get(playing id)
071                           .getMusicSinger());
072
073             //更新歌曲时间
074             MusicPlayerActivity.time right.setText(Music infoAdapter
075                     .toTime(Music infoAdapter.musicList.get(playing id)
076                           .getMusicTime()));
077
078             // 获取专辑图片路径
079             String url = MusicPlayerActivity.music info
080                     .getAlbumArt(Music infoAdapter.musicList.get(
081                           ControlPlay.playing id).get id());
082             if (url != null) {
083
084                 //显示获取的专辑图片
085                 MusicPlayerActivity.music_AlbumArt.setImageURI(Uri
086                         .parse(url));
087                 MusicPlayerActivity.music_AlbumArt
088                         .setAnimation(AnimationUtils.loadAnimation(
089                                 MusicPlayerActivity.context,
090                                 R.anim.alpha z));
091
092                 //开启通知，传入歌曲名和艺术家及通知图标
093                 MusicPlayerActivity.setNotice(c, c,
094                         Music_infoAdapter.musicList.get(playing_id)
095                               .getMusicSinger(), R.drawable.notice_icon);
096
097                 try {
098                     /* 为倒影创建位图 */
099                     Bitmap mBitmap = BitmapFactory.decodeFile(url);
```

```
100                     MusicPlayerActivity.reflaction
101                             .setImageBitmap(MusicPlayerActivity
102                                     .createReflectedImage(mBitmap));
103                     MusicPlayerActivity.reflaction
104                             .setAnimation(AnimationUtils.loadAnimation(
105                                     MusicPlayerActivity.context,
106                                     R.anim.alpha z));
107                 } catch (Exception e) {
108                     // TODO Auto-generated catch block
109                     e.printStackTrace();
110                 }
111 
112             } else {
113                 MusicPlayerActivity.music_AlbumArt
114                         .setImageResource(R.drawable.album);
115                 MusicPlayerActivity.music_AlbumArt
116                         .setAnimation(AnimationUtils.loadAnimation(
117                                 MusicPlayerActivity.context,
118                                 R.anim.alpha_y));
119 
120                 //开启通知，传入歌曲名和艺术家及通知图标
121                 MusicPlayerActivity.setNotice(c, c,
122                         Music infoAdapter.musicList.get(playing id)
123                                 .getMusicSinger(),
                                 R.drawable.notice_icon);
124 
125                 try {
126                     /* 为倒影创建位图 */
127                     Bitmap mBitmap = ((BitmapDrawable) getResources()
128                             .getDrawable(R.drawable.album)).getBitmap();
129                     MusicPlayerActivity.reflaction
130                             .setImageBitmap(MusicPlayerActivity
131                                     .createReflectedImage(mBitmap));
132                     MusicPlayerActivity.reflaction
133                             .setAnimation(AnimationUtils.loadAnimation(
134                                     MusicPlayerActivity.context,
135                                     R.anim.alpha z));
136                 } catch (Exception e) {
137                     // TODO Auto-generated catch block
138                     e.printStackTrace();
139                 }
140             }
141         }
142 
143         /**
144          * 监听播放是否完毕
145          */
146         myMediaPlayer.setOnCompletionListener(new OnComp
        letionListener() {
147 
148             @Override
149             public void onCompletion(MediaPlayer mp) {
150                 // TODO Auto-generated method stub
151 
152                 //播放完当前歌曲，自动播放下一首
153                 playNext();
154             }
155         });
156 
157     } else {
```

```
158         Toast.makeText(ControlPlay.this, "没有在手机里找到歌曲...",
159                 Toast.LENGTH_SHORT).show();
160     }
161 }
162
163 /**
164  * 播放下一首
165  */
166 public void playNext() {
167
168     //判断歌曲不能为空
169     if (Music_infoAdapter.musicList.size() != 0) {
170         //如果到了最后一首则一直播放最后一首
171         if (playing_id == Music_infoAdapter.musicList.size() - 1) {
172             playing_id = Music_infoAdapter.musicList.size() - 1;
173             Toast.makeText(ControlPlay.this, "已经是最后一首啦！",
174                     Toast.LENGTH_SHORT).show();
175
176             MusicPlayerActivity.play_ImageButton
177                     .setBackgroundResource(R.drawable.play_button);
178             MusicPlayerActivity.mNotificationManager.cancel(1);
179
180         } else {
181             initMediaSource(initMusicUri(++playing_id));
182             playMusic();
183         }
184     } else {
185         Toast.makeText(ControlPlay.this, "没有找到歌曲！", Toast.LENGTH_
        SHORT)
186                 .show();
187     }
188 }
189
190 /**
191  * 播放上一首
192  */
193 public void playFront() {
194
195     //判断歌曲不能为空
196     if (Music_infoAdapter.musicList.size() != 0) {
197         //如果到了第一首则一直播放第一首
198         if (playing_id == 0) {
199             playing_id = 0;
200             Toast.makeText(ControlPlay.this, "现在就是第一首哦！",
201                     Toast.LENGTH_SHORT).show();
202         } else {
203             initMediaSource(initMusicUri(--playing_id));
204             playMusic();
205         }
206     } else {
207         Toast.makeText(ControlPlay.this, "没有找到歌曲啊！
        ", Toast.LENGTH_SHORT)
208                 .show();
209     }
210 }
```

歌曲播放类最后一部分是歌词的处理部分，代码如下所示，分为歌曲进度条更新部分和歌词滚动部分。歌曲进度条的更新部分在 updatesb 中实现，通过获得音乐当前播放的时

刻，更新音乐进度条的位置。歌词滚动部分在 mRunnable 中实现，通过函数 LrcIndex()取得当前音乐对应的歌词的位置，进而高亮显示对应的歌词。

```
01  public void startSeekBarUpdate() {
02      // TODO Auto-generated method stub
03      handler.post(start);
04  }
05
06  Runnable start = new Runnable() {
07
08      @Override
09      public void run() {
10          // TODO Auto-generated method stub
11
12          handler.post(updatesb);
13      }
14
15  };
16
17  Runnable updatesb = new Runnable() {
18
19      @Override
20      public void run() {
21          //TODO Auto-generated method stub
22
23          //获取 SeekBar 走动到那的时间
24          MusicPlayerActivity.play_time = myMediaPlayer
25                  .getCurrentPosition();
26
27          //设置填充当前获取的进度
28          MusicPlayerActivity.seekbar
29                  .setProgress(MusicPlayerActivity.play_time);
30          //SeekBar 的最大值填充歌曲时间
31          MusicPlayerActivity.seekbar.setMax(Music_infoAdapter
          .musicList
32                  .get(playing_id).getMusicTime());
33
34          //线程延迟 1000 毫秒启动
35          handler.postDelayed(updatesb, 1000);
36      }
37  };
38
39  Handler mHandler = new Handler();
40  //歌词滚动线程
41  Runnable mRunnable = new Runnable() {
42
43      @Override
44      public void run() {
45          // TODO Auto-generated method stub
46          MusicPlayerActivity.lrc_view.SetIndex(LrcIndex());
47          MusicPlayerActivity.lrc_view.invalidate();
48          mHandler.postDelayed(mRunnable, 100);
49      }
50  };
51
52  /**
53   * 歌词同步处理类
54   */
55  public int LrcIndex() {
```

```
56      if (myMediaPlayer.isPlaying()) {
57          //获得歌曲播放到哪的时间
58          CurrentTime = myMediaPlayer.getCurrentPosition();
59          //获得歌曲总时间长度
60          CountTime = myMediaPlayer.getDuration();
61      }
62      if (CurrentTime < CountTime) {
63
64          for (int i = 0; i < lrcList.size(); i++) {
65              if (i < lrcList.size() - 1) {
66                  if (CurrentTime < lrcList.get(i).getLrc time() && i == 0)
                    {
67                      index = i;
68                  }
69                  if (CurrentTime > lrcList.get(i).getLrc time()
70                          && CurrentTime < lrcList.get(i + 1).getLrc time()
                          ) {
71                      index = i;
72                  }
73              }
74              if (i == lrcList.size() - 1
75                      && CurrentTime > lrcList.get(i).getLrc_time()) {
76                  index = i;
77              }
78          }
79      }
80      return index;
81  }
```

17.7　知 识 拓 展

我们在音乐播放服务类里面使用了 handler.post()来更新界面，下面我们用一个小例子来进一步分析这种更新界面的方式。

如下所示，我们在 onCreate()函数中调用 mHandler.post(update)来更新界面，值得注意的是，这个 Handler 是在主线程中创建的，也就是说这个 Handler 与主线程在同一个线程。因此在 update 中我们不能做耗时的动作如下载等，这样会导致界面停止响应。言归正传，我们通过在 run 中调用 postDelayed(update,5)来递归调用 update()函数，同时在 update()函数中调用 postInvalidate()来使当前视图失效，进而重新调用 onDraw 来更新界面。

```
public class HandlePostActivity extends Activity {
    private MyView myView;
    private Handler mHandler;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        myView = new MyView(this);
        mHandler = new Handler();
        mHandler.post(update);
        setContentView(myView);
    }
    private Runnable update = new Runnable() {
        public void run() {
            myView.update();
            mHandler.postDelayed(update, 5);
        }
```

```
    };
    //自定义视图类
    class MyView extends View{
        private float x = 0f;
        public MyView(Context context) {
            super(context);

        }
        //使当前视图无效，从而重新调用 onDraw
        public void update(){
            postInvalidate();
        }
        //更新视图
        protected void onDraw(Canvas canvas) {
            super.onDraw(canvas);
            x+=1;
            Paint mPaint = new Paint();
            mPaint.setColor(Color.BLUE);
            //绘制矩形
            canvas.drawRect(x, 40, x+40, 80, mPaint);
}
```

17.8　本章小结

本章介绍了 Android MP3 播放器的开发，界面开发采用了模仿 Android 桌面的设计，而在处理音乐文件时则采用了系统自带的 MediaPlayer 类。通过本章的学习，读者需要掌握如何设计一个可以“滑动”的界面，学会灵活运用 Android 的界面设计方法，设计出既高效又美观的界面。同时，读者要深入学习 MediaPlayer 类的使用，了解每一个函数的用法，在今后自己的程序中能够灵活运用该类处理音频文件。

第 18 章　Android 照相机

Android 手机一般都要配备高像素的摄像头，利用手机自带的照相软件可以随时捕捉您想要的精彩瞬间。那么 Android 照相机该如何调用呢？如何在我们自己的程序中使用照相机呢？本章我们将制作一个小程序，用来演示 Android 照相机的调用。

18.1　调用 Android 相机的两种方式

Android 手机关于 Camera 的使用有两种，一是拍照，二是摄像。Android 提供了强大的组件功能，在 Android 手机系统上进行 Camera 的开发，我们可以使用两类方法：一是借助 Intent 和 MediaStore 调用系统 Camera App 程序来实现拍照和摄像功能，二是根据 Camera API 自写 Camera 程序。自写 Camera 需要对 Camera API 了解很充分，而对于通用的拍照和摄像应用只需要借助系统 Camera App 程序就能满足要求了。

18.1.1　调用系统自带相机

（1）要调用系统相机，只需要使用 Intent 启动相机，代码如下所示：

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, 1);
```

（2）拍完照后就可以在 onActivityResult(int requestCode, int resultCode, Intent data)中获取到 Bitmap 对象了。

```
Bitmap bitmap = (Bitmap) data.getExtras().get("data");
```

（3）获得了图像之后可以将图像存储到 SD 中，在存储到 SD 前最好先检查一下 SD 是否可用，代码如下所示：

```
String sdStatus = Environment.getExternalStorageState();
if (!sdStatus.equals(Environment.MEDIA_MOUNTED)) {
    // 检测 SD 是否可用
    Log.v("TestFile","SD card is not avaiable/writeable right now.");
    return;
}
```

（4）检查完 SD 卡的可用性之后，就可以将图像文件存到 sdcard/picture/文件夹下，名称为 test.jpg。

代码如下所示：

```
01  File file = new File("/sdcard/picture/");
```

```
02  //创建文件夹
03  file.mkdirs();
04  String fileName = "/sdcard/picture/test.jpg";
05  try {
06      b = new FileOutputStream(fileName);
07      //把数据写入文件
08      bitmap.compress(Bitmap.CompressFormat.JPEG, 100, b);
09  } catch (FileNotFoundException e) {
10      e.printStackTrace();
11  } finally {
12      try {
13          b.flush();
14          b.close();
15      } catch (IOException e) {
16          e.printStackTrace();
17      }
18  }
```

（5）另外要注意的是读写 SD 卡文件必须首先要在 Manifest.xml 文件中配置权限：

```
<uses-permission android:name="android.permission.WRITE EXTERNAL
STORAGE" />
<uses-permission android:name="android.permission.MOUNT UNMOUNT FILESYS
TEMS" />
```

18.1.2　根据 Camera API 实现自己的拍照程序

（1）上面调用了系统 Camera App，我们不需要任何权限，但是这里调用 Camera API，就必须在 Manifest.xml 内声明使用权限，通常由以下 3 项组成：

```
<uses-permission android:name = "android.permission.CAMERA" />
<uses-feature android:name = "android.hardware.camera" />
<uses-feature android:name = "android.hardware.camera.autofocus" />
```

（2）一般拍照的时候需要将照片保存到 SD 卡上，所以还有一项权限声明如下：

```
uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

（3）在 Activity 的 OnCreate 函数中设置好 SurfaceView，包括设置 SurfaceHolder.Callback 对象和 SurfaceHolder 对象的类型，如下所示：

```
SurfaceView mpreview = (SurfaceView) this.findViewById(R.id.
camera_preview);
SurfaceHolder mSurfaceHolder = mpreview.getHolder();
mSurfaceHolder.addCallback(this);
mSurfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

（4）在 SurfaceHolder.Callback 的 surfaceCreated()函数中，使用 Camera 的 open()函数开启摄像头硬件，若开启成功则返回一个 Camera 对象，否则就抛出异常。在开启成功的情况下，在 SurfaceHolder.Callback 的 surfaceChanged()函数中调用 getParameters()函数得到当前打开的摄像头的配置参数 Parameters 对象，如果需要就修改对象的参数，然后调用 setParameters()函数设置进去。

同样在 surfaceChanged()函数中，调用 setPreviewDisplay 为摄像头设置 SurfaceHolder

对象，设置成功后调用 startPreview()函数开启预览功能，如下代码所示：

```
01    @Override
02    public void surfaceCreated(SurfaceHolder holder) {
03        // TODO Auto-generated method stub
04        //开启相机
05        if(camera == null)
06        {
07            camera = Camera.open();
08            try {
09                camera.setPreviewDisplay(holder);
10            } catch (IOException e) {
11                // TODO Auto-generated catch block
12                e.printStackTrace();
13            }
14        }
15
16    }
17  public void surfaceChanged(SurfaceHolder holder, int format, int w,
    int h)
18  {
19      //已经获得 Surface 的 width 和 height，设置 Camera 的参数
20      Camera.Parameters parameters = camera.getParameters();
21      parameters.setPreviewSize(w, h);
22      List<Size> vSizeList = parameters.getSupportedPictureSizes();
23      for(int num = 0; num < vSizeList.size(); num++)
24      {
25      Size vSize = vSizeList.get(num);
26      }
27          if(this.getResources().getConfiguration().orientation
            != Configuration.ORIENTATION LANDSCAPE)
28      {
29          //如果是竖屏
30          parameters.set("orientation", "portrait");
31      }
32      else
33      {
34          parameters.set("orientation", "landscape");
35      }
36      camera.setParameters(parameters);
37      try {
38      //设置显示
39          camera.setPreviewDisplay(holder);
40      } catch (IOException exception) {
41          camera.release();
42          camera = null;
43      }
44      //开始预览
45      camera.startPreview();
46  }
```

（5）如果要支持自动对焦功能，则在需要的情况下，或者在上述 surfaceChanged 调用完 startPreview()函数后，可以调用 Camera 的 autoFocus()函数来设置自动对焦回调函数。该步是可选操作，有些设备可能不支持，可以通过 Camera 的 getFocusMode()函数查询。

```
01  //自动对焦
02  camera.autoFocus(new AutoFocusCallback()
03  {
04  @Override
```

```
05  public void onAutoFocus(boolean success, Camera camera)
06  {
07      if (success)
08      {
09          // success 为 true 表示对焦成功，改变对焦状态图像
10          ivFocus.setImageResource(R.drawable.focus2);
11      }
12  }
13  });
```

（6）在需要拍照的时候，调用 takePicture(Camera.ShutterCallback, Camera.PictureCallback, Camera.PictureCallback, Camera.PictureCallback)函数来完成拍照。这个函数中有 4 个回调接口，ShutterCallback 是快门按下的回调，在这里我们可以设置播放“咔嚓”声之类的操作，后面有 3 个 PictureCallback 接口，对应 3 份图像数据，分别是原始图像、缩放和压缩图像及 JPG 图像，图像数据可以在 PictureCallback 接口的 void onPictureTaken(byte[] data, Camera camera)中获得。3 时份数据相应的 3 个回调正好按照参数顺序调用，通常我们只关心 JPG 图像数据，此时前面两个 PictureCallback 接口参数可以直接传 null。

每次调用 takePicture 获取图像后，摄像头会停止预览。假如需要继续拍照，则我们需要在上面的 PictureCallback 的 onPictureTaken()函数末尾，再次调用 Camera 的 startPreview()函数。

在不需要拍照的时候，我们需要主动调用 Camera 的 stopPreview()函数停止预览功能，并且调用 Camera 的 release()函数释放 Camera，以便其他应用程序调用。

```
1   //停止拍照时调用该方法
2   public void surfaceDestroyed(SurfaceHolder holder)
3   {
4       //释放手机摄像头
5       camera.release();
6   }
```

18.2　相机界面设计

接下来我们要设计一个简易相机，在 res/layout 下新建 main.xml，代码如下所示。整个界面由两部分组成，上面是由一个 SurfaceView 构成的相机预览界面，设置界面固定高度为 420dp，下面是一个拍照按键和一个 ImageView 用于显示拍照生成的缩略图。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout
03      xmlns:android="http://schemas.android.com/apk/res/android"
04      android:orientation="vertical"
05      android:layout width="fill parent"
06      android:layout_height="fill_parent"
07      >
08      <!-- 相机预览界面 -->
09      <SurfaceView
10          android:id="@+id/surfaceView1"
11          android:layout width="fill parent"
12          android:layout height="420dp"
13          android:layout_gravity="center">
```

```
14          <!-- 用户操作区 -->
15      </SurfaceView>
16      <LinearLayout
17          android:orientation="horizontal"
18          android:layout_width="fill_parent"
19          android:layout_height="fill_parent"
20          android:layout_marginTop="20dp"
21          >
22          <!-- 拍照按键 -->
23          <Button
24              android:id="@+id/capture"
25              android:layout width="100dp"
26              android:layout height="100dp"
27              android:text="拍照"/>
28          <!-- 生成的照片缩略图 -->
29          <ImageView
30              android:layout marginLeft="50dp"
31              android:id="@+id/editPic"
32              android:layout width="100dp"
33              android:layout height="100dp"
34              android:contentDescription="照片"
35               />
36      </LinearLayout>
37  </LinearLayout>
```

当用户单击生成的缩略图，可以进一步查看刚才拍摄的照片。代码如下所示，在 res/layout 下新建 picture.xml，上面一个 ImageView 用于显示照片，下面一个放大缩小控件用于对图片进行放大缩小。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com
     /apk/res/android"
03      android:id="@+id/layout1"
04      android:layout width="match parent"
05      android:layout height="match parent"
06      android:orientation="vertical" >
07      <!-- 图片查看区 -->
08      <ImageView
09          android:id="@+id/img"
10          android:scaleType="center"
11          android:layout_width="wrap_content"
12          android:layout_height="420dp" />
13      <!—放大缩小控件 -->
14      <ZoomControls
15          android:id="@+id/zoomControls1"
16          android:layout width="wrap content"
17          android:layout height="wrap content"
18          android:layout gravity="bottom|center" />
19  </LinearLayout>
```

18.3　相机功能实现

介绍完界面的实现，下面我们来分析一下相应的功能实现。

18.3.1　拍照功能实现

（1）新建 CameraTest.java，用于实现拍照功能，在该文件中新建类 CameraTest 继承于 Activity 并实现 Callback 和 AutoFocusCallback 接口。在类的开始，首先需要声明一些需要用到的成员变量如下所示：

```
01 //surfaceView 声明
02   SurfaceView mySurfaceView;
03   //surfaceHolder 声明
04   SurfaceHolder holder;
05   //相机声明
06   Camera myCamera;
07   //照片保存路径
08   String filePath="/sdcard/Pictures/";
09   //是否单击标志位
10   boolean isClicked = false;
11   //拍照按钮
12   Button capture;
13   //照片缩略图
14   ImageView editPic;
15   Context mContext;
```

（2）如下所示，在初始化函数中获得 SurfaceView 的句柄 holder，并添加回调函数，其中很关键的一点是要设置 SurfaceView 的类型，没有这句的话将会导致摄像头调用出错，但是 Android 的 SDK 中有说明这句可以省略，让人摸不着头脑。接着获得界面元素，并为它们设置监听器。

```
01   @Override
02   public void onCreate(Bundle savedInstanceState) {
03       super.onCreate(savedInstanceState);
04       //无标题
05       requestWindowFeature(Window.FEATURE_NO_TITLE);
06       //设置拍摄方向
07       this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION
         _SENSOR_
         PORTRAIT);
08       setContentView(R.layout.main);
09       //获得控件
10       mySurfaceView = (SurfaceView)findViewById(R.id.surfaceView1);
11       //获得句柄
12       holder = mySurfaceView.getHolder();
13       //添加回调
14       holder.addCallback(this);
15       mContext=this;
16       //设置类型，没有这句将调用失败
17       holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
18       //拍照按钮
19       capture=(Button)findViewById(R.id.capture);
20       //缩略图
21       editPic=(ImageView)findViewById(R.id.editPic);
22       //设置按键监听器
23       capture.setOnClickListener(takePicture);
24       editPic.setOnClickListener(editOnClickListener);
```

```
25    }
```

（3）SurfaceView 初始化过程中将依次执行 surfaceCreated()函数和 surfaceChanged()函数，我们在 surfaceCreated()函数中调用 Camera.open()获得照相机实例，并在srufaceChanged 中设置摄像头的参数。由于摄像头预览的方向与实际方向相差了 90°，故我们将显示方向旋转 90°，设置完参数之后，开始预览。

在 surfaceDestroyed 中我们需要调用 stopPreview 和 release 来关闭预览并释放资源。

```
01  @Override
02    public void surfaceChanged(SurfaceHolder holder, int format,int width,
03            int height) {
04        // TODO Auto-generated method stub
05        //设置参数
06        Camera.Parameters params = myCamera.getParameters();
07        params.setPictureFormat(PixelFormat.JPEG);
08        myCamera.setParameters(params);
09        //设置预览方向旋转 90°
10        myCamera.setDisplayOrientation(90);
11        //开始预览
12        myCamera.startPreview();
13
14    }
15    @Override
16    public void surfaceCreated(SurfaceHolder holder) {
17        // TODO Auto-generated method stub
18        //开启相机
19        if(myCamera == null)
20        {
21            myCamera = Camera.open();
22            try {
23                myCamera.setPreviewDisplay(holder);
24            } catch (IOException e) {
25                // TODO Auto-generated catch block
26                e.printStackTrace();
27            }
28        }
29
30    }
31    @Override
32    public void surfaceDestroyed(SurfaceHolder holder) {
33        // TODO Auto-generated method stub
34        //关闭预览并释放资源
35        myCamera.stopPreview();
36        myCamera.release();
37        myCamera = null;
38
39    }
```

（4）当我们单击拍照按钮时，将调用自动对焦函数，在对焦函数中首先设置照片格式为 JPEG，接着调用 takePicture 进行拍照。拍照的整个处理过程在函数 jpeg 中实现。代码如下所示，首先调用函数 BitmapFactory.decodeByteArray()将数据转换成 Bitmap。因为图片方向与实际方向相差 90°，因此我们需要在保存之前将图片旋转 90°，再将数据保存到 SD 卡中。

同时我们调用函数 changeBitmapToDrawable()将图片进行缩放并显示到界面相应位置上。

```
01 //拍照按键监听器
02   OnClickListener takePicture=new OnClickListener(){
03     @Override
04     public void onClick(View v) {
05         // TODO Auto-generated method stub
06         if(!isClicked)
07         {
08             //自动对焦
09             myCamera.autoFocus(CameraTest.this);
10             isClicked = true;
11         }else
12         {
13             //开启预览
14             myCamera.startPreview();
15             isClicked = false;
16         }
17     }
18 };
19   //自动对焦时调用
20   @Override
21   public void onAutoFocus(boolean success, Camera camera) {
22       // TODO Auto-generated method stub
23       if(success)
24       {
25           //获得参数
26           Camera.Parameters params = myCamera.getParameters();
27           //设置参数
28           params.setPictureFormat(PixelFormat.JPEG);
29           myCamera.setParameters(params);
30           //拍照
31           myCamera.takePicture(null, null, jpeg);
32       }
33   }
34   PictureCallback jpeg = new PictureCallback() {
35       @Override
36       public void onPictureTaken(byte[] data, Camera camera) {
37           // TODO Auto-generated method stub
38           try
39           {
40           //获得图片
41           Bitmap bm = BitmapFactory.decodeByteArray(data,
          0, data.length);
42           SimpleDateFormat sDateFormat = new SimpleDateFormat
          ("yyyyMMddhhmmss");
43           String date = sDateFormat.format(new java.util.Date());
44           filePath=filePath+date+".jpg";
45           File file = new File(filePath);
46           BufferedOutputStream bos = new BufferedOutputStream
          (new FileOutputStream(file));
47           //创建操作图片用的 matrix 对象
48          Matrix matrix = new Matrix();
49          matrix.postRotate(90);
50          //创建新的图片
51          Bitmap rotateBitmap = Bitmap.createBitmap(bm, 0,0,
           bm.getWidth(), bm.getHeight(), matrix, true);
52           //将图片以 JPEG 格式压缩到流中
53         rotateBitmap.compress(Bitmap.CompressFormat.JPEG, 100, bos);
54         //输出
55           bos.flush();
```

```
56              //关闭
57              bos.close();
58              editPic.setBackgroundDrawable(changeBitmapToDrawable
                (rotateBitmap));
59              editPic.setTag(filePath);
60              }catch(Exception e)
61              {
62                  e.printStackTrace();
63              }
64          }
65      };
66      public BitmapDrawable changeBitmapToDrawable(Bitmap bitmapOrg)
67      {
68         int width = bitmapOrg.getWidth();
69         int height = bitmapOrg.getHeight();
70
71         //定义想要转换成的图片的宽和高
72         int newWidth = 100;
73
74         //计算缩放率，新尺寸除原尺寸
75         float scaleWidth = (float)newWidth/width;
76         float scaleHeight = scaleWidth;
77         //创建操作图片用的 matrix 对象
78         Matrix matrix = new Matrix();
79         //缩放图片动作
80         matrix.postScale(scaleWidth, scaleHeight);
81         //创建新的图片
82         Bitmap resizedBitmap = Bitmap.createBitmap(bitmapOrg, 0, 0,width,
           height, matrix, true);
83         //将上面创建的 Bitmap 转换成 Drawable 对象，使得其可以使用在
            imageView、imageButton 上
84         BitmapDrawable bitmapDrawable = new BitmapDrawable(resizedBitmap);
85         return bitmapDrawable;
86      }
```

（5）当我们单击图片缩略图时，将调用一下监听器，启动查看图片界面并关闭当前界面。

```
01      //查看图片
02      OnClickListener editOnClickListener=new OnClickListener(){
03        @Override
04        public void onClick(View v) {
05            // TODO Auto-generated method stub
06            String picPath=(String) v.getTag();
07            Intent intent=new Intent();
08            //将图片的路径绑定到 intent 中
09            intent.putExtra("path", picPath);
10            intent.setClass(mContext, Picture.class);
11            //启动查看图片界面
12            startActivity(intent);
13            //关闭当前界面
14            CameraTest.this.finish();
15        }
16    };
```

18.3.2　照片查看

如下所示新建 Picture 类继承于 Activity，在 onCreate()函数中通过传递过来的图片的路

径获得图片，并获得当前屏幕显示区域的高宽和图片的高宽，接着为放大缩小控件设定按键监听器。

当单击放大按钮时，图片将以 1.25 倍的速率放大，为了防止图片过大超过内存，在此设置了一个极限值 1.25 倍。当单击缩小按钮时，图片将以 0.8 倍的速率缩小。如图 18.1 所示为图片查看界面的截图。



图 18.1　图片查看界面

```
01  public class Picture extends Activity {
02       //图片
03       ImageView iv;
04       //放大缩小控制器
05       ZoomControls zoom;
06       //屏幕显示区域宽度
07       private int displayWidth;
08       //屏幕显示区域高度
09       private int displayHeight;
10       private float scaleWidth = 1;
11       private float scaleHeight = 1;
12       //图片宽度
13       int bmpWidth;
14       //图片高度
15       int bmpHeight;
16       Bitmap bitmapOrg;
17       @Override
18       protected void onCreate (Bundle savedInstanceState) {
19       super.onCreate(savedInstanceState);
20       setContentView(R.layout.picture);
21       zoom=(ZoomControls)findViewById(R.id.zoomControls1);
22       //取得屏幕分辨率大小
23       DisplayMetrics dm = new DisplayMetrics();
```

```
24         getWindowManager().getDefaultDisplay().getMetrics(dm);
25         displayWidth = dm.widthPixels;
26         //屏幕高度减去 zoomControls 的高度
27         displayHeight = dm.heightPixels-80;
28         zoom.setIsZoomInEnabled(true);
29         zoom.setIsZoomOutEnabled(true);
30
31         //创建一个 ImageView
32         iv = (ImageView)findViewById(R.id.img);
33         Intent it=getIntent();
34         String picPath=(String) it.getCharSequenceExtra("path");
35         bitmapOrg=BitmapFactory.decodeFile(picPath,null);
36         iv.setImageBitmap(bitmapOrg);
37         //获得图片的高度和宽度
38         bmpWidth = bitmapOrg.getWidth();
39         bmpHeight = bitmapOrg.getHeight();
40         //图片放大
41        zoom.setOnZoomInClickListener(new OnClickListener()
42        {
43           public void onClick(View v)
44           {
45               //设置图片放大的比例
46               double scale = 1.25;
47               //计算这次要放大的比例
48               scaleWidth = (float)(scaleWidth*scale);
49               scaleHeight = (float)(scaleHeight*scale);
50               if(scaleWidth > 1.25)
51               {
52                  scaleWidth=1;
53                  scaleHeight=1;
54               }
55               //产生新的大小的 Bitmap 对象
56               Matrix matrix = new Matrix();
57               matrix.postScale(scaleWidth, scaleHeight);
58               Bitmap resizeBmp = Bitmap.createBitmap(bitmapOrg,0,
                 0,bmpWidth,bmpHeight,matrix,true);
59               iv.setImageBitmap(resizeBmp);
60           }
61        });
62        //图片减小
63        zoom.setOnZoomOutClickListener(new OnClickListener(){
64               public void onClick(View v) {
65               //设置图片放大的比例
66               double scale = 0.8;
67               //计算这次要放大的比例
68               scaleWidth = (float)(scaleWidth*scale);
69               scaleHeight = (float)(scaleHeight*scale);
70               //产生新的大小的 Bitmap 对象
71               Matrix matrix = new Matrix();
72               matrix.postScale(scaleWidth, scaleHeight);
73               Bitmap resizeBmp = Bitmap.createBitmap(bitmapOrg,0,0,
                 bmpWidth,bmpHeight,matrix,true);
74               iv.setImageBitmap(resizeBmp);
75           }
76        });
77        }
```

最后为该界面添加菜单项，一个为“返回”，用于返回拍照界面；另一个为“退出”，

用于退出当前程序。

```
01  //添加菜单选项
02  public boolean onCreateOptionsMenu(Menu menu)
03  {
04      super.onCreateOptionsMenu(menu);
05      //返回
06      menu.add(0, 1, 0, "返回");
07      //退出
08      menu.add(0, 2, 0, "退出");
09      return true;
10  }
11  //处理菜单操作
12  public boolean onOptionsItemSelected(MenuItem item)
13  {
14      switch (item.getItemId())
15      {
16      case 1:
17          //返回拍照界面
18          Intent intent=new Intent();
19          intent.setClass(this, CameraTest.class);
20          startActivity(intent);
21          this.finish();
22          return true;
23      case 2:
24          //退出程序
25          this.finish();
26          return true;
27      }
28      return super.onOptionsItemSelected(item);
29  }
```

18.4　知识拓展

在 Android 中，Matrix 的操作总共分为 translate（平移）、rotate（旋转）、scale（缩放）和 skew（倾斜）4 种，每一种变换在 Android 的 API 里都提供了 set、post 和 pre 3 种操作方式，除了 translate，其他 3 种操作都可以指定中心点。

其中，set 是直接设置 Matrix 的值，每次 set 一次，整个 Matrix 的数组都会变掉。其次 post 是后乘，当前的矩阵乘以参数给出的矩阵。可以连续多次使用 post，来完成所需的整个变换。例如，要将一个图片旋转 30°，然后平移到(100,100)的地方，可以这样做：

```
Matrix m = new Matrix();
m.postRotate(30);
m.postTranslate(100, 100);
Matrix m = new Matrix();
m.postRotate(30);
m.postTranslate(100, 100);
```

最后 pre 是前乘，参数给出的矩阵乘以当前的矩阵，所以操作是在当前矩阵的最前面发生的。例如上面的例子，如果用 pre 的话，可以这样做：

```
Matrix m = new Matrix();
```

```
m.setTranslate(100, 100);
m.preRotate(30);
```

旋转、缩放和倾斜都可以围绕一个中心点来进行，如果不指定，默认情况下，是围绕(0,0)点来进行。

18.5　本 章 小 结

本章介绍了调用相机的两种方式，并以介绍自制相机程序的开发过程，来进一步讲述如何通过调用系统 Camera 相关 API 实现自己的相机。本章的目的是为了让读者学会在自己的程序中嵌入相机的应用，如微博分享等，丰富程序的功能。

第 19 章　视频播放器

Android 平台上有多种视频播放器，各有特色，功能都很强大。这些播放器有些是单一播放器，比如专为播放 SWF 格式开发的播放器，但大多数播放器都支持多种视频格式。本章将开发一款视频播放器，该视频播放器的解码方式基于系统本身的 MediaPlayer 库。

19.1　视频播放界面设计

图 19.1 所示为视频播放器的主界面，和大多数播放器的界面类似，上面为播放界面，下面的视频控制条作为弹出窗口可以显示或隐藏。用户可以通过视频控制条来控制视频的播放，包括视频进度的控制、视频源的选择、播放暂停功能的切换、上一个视频和下一个视频的选择、音量大小的控制等等。

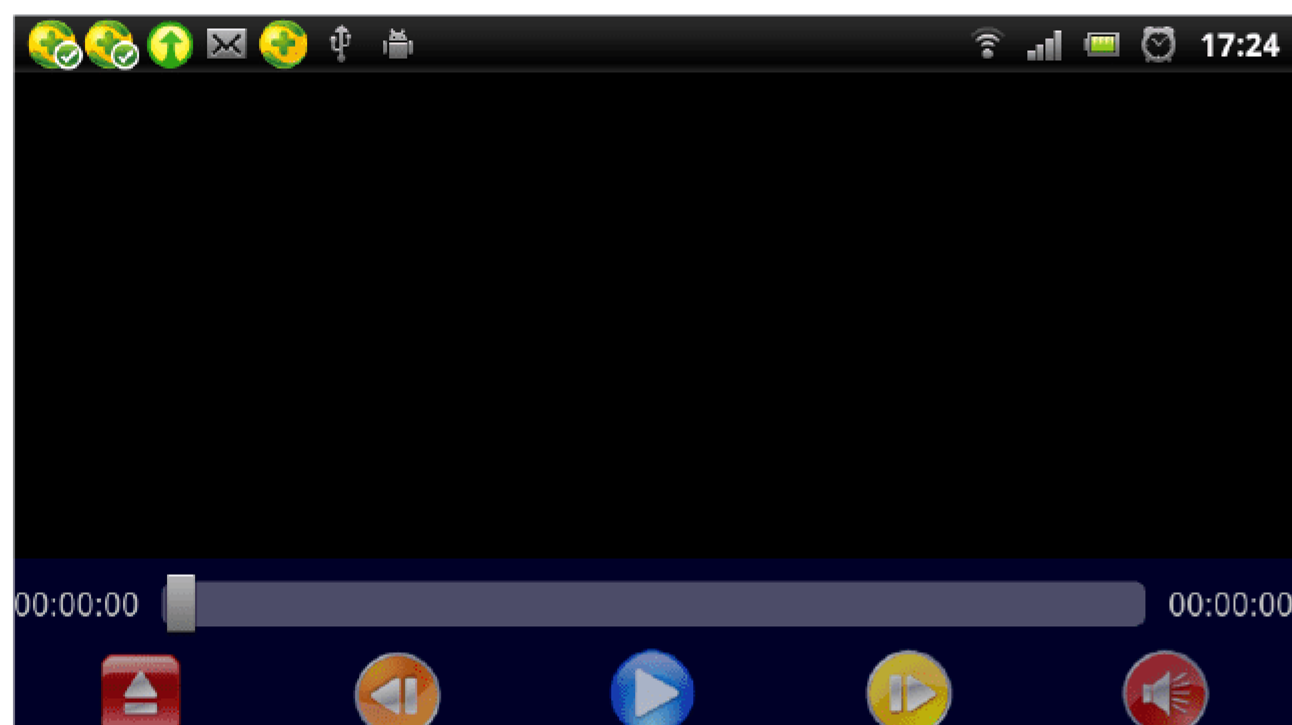


图 19.1　视频播放主界面

用户打开视频播放器时，可以通过左下角的“弹出”按钮，显示当前 SD 卡目录中的所有满足条件的视频文件，从中选择想要播放的视频，如图 19.2 所示。

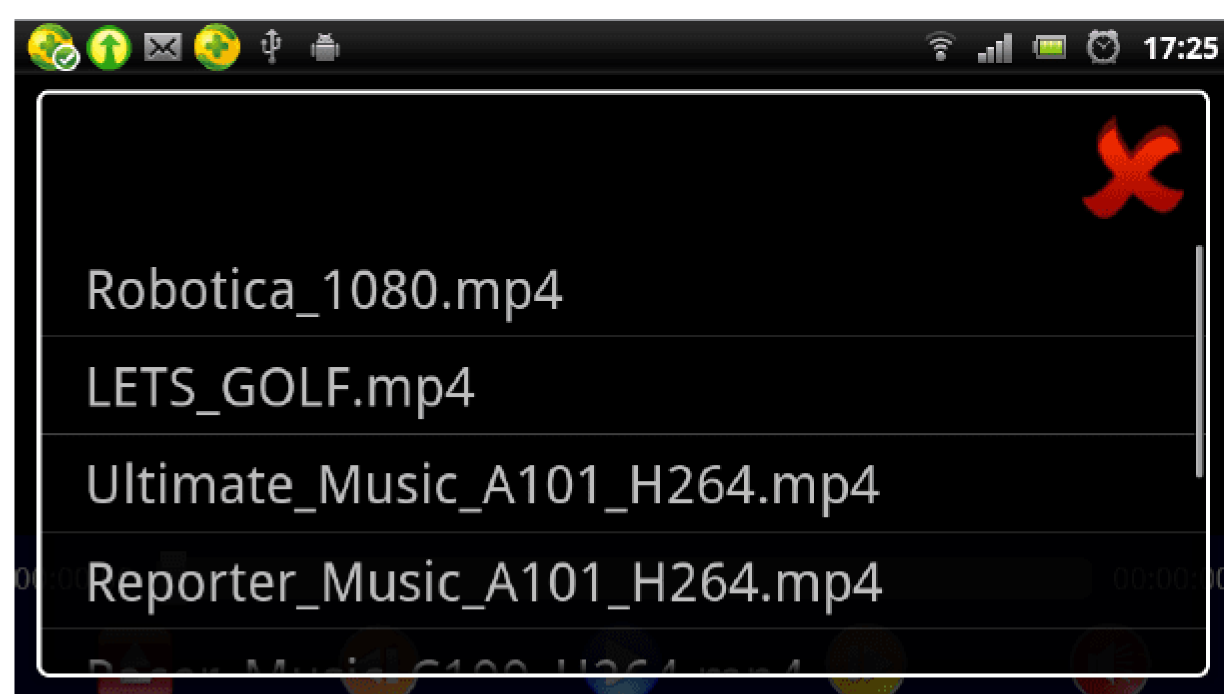


图 19.2　视频文件选择界面

除此之外，我们还设计了另一个界面——音量控制界面，当用户单击控制条中的音量图标时将会弹出音量控制界面，如图 19.3 所示，通过上下触摸可以增大或减小音量。

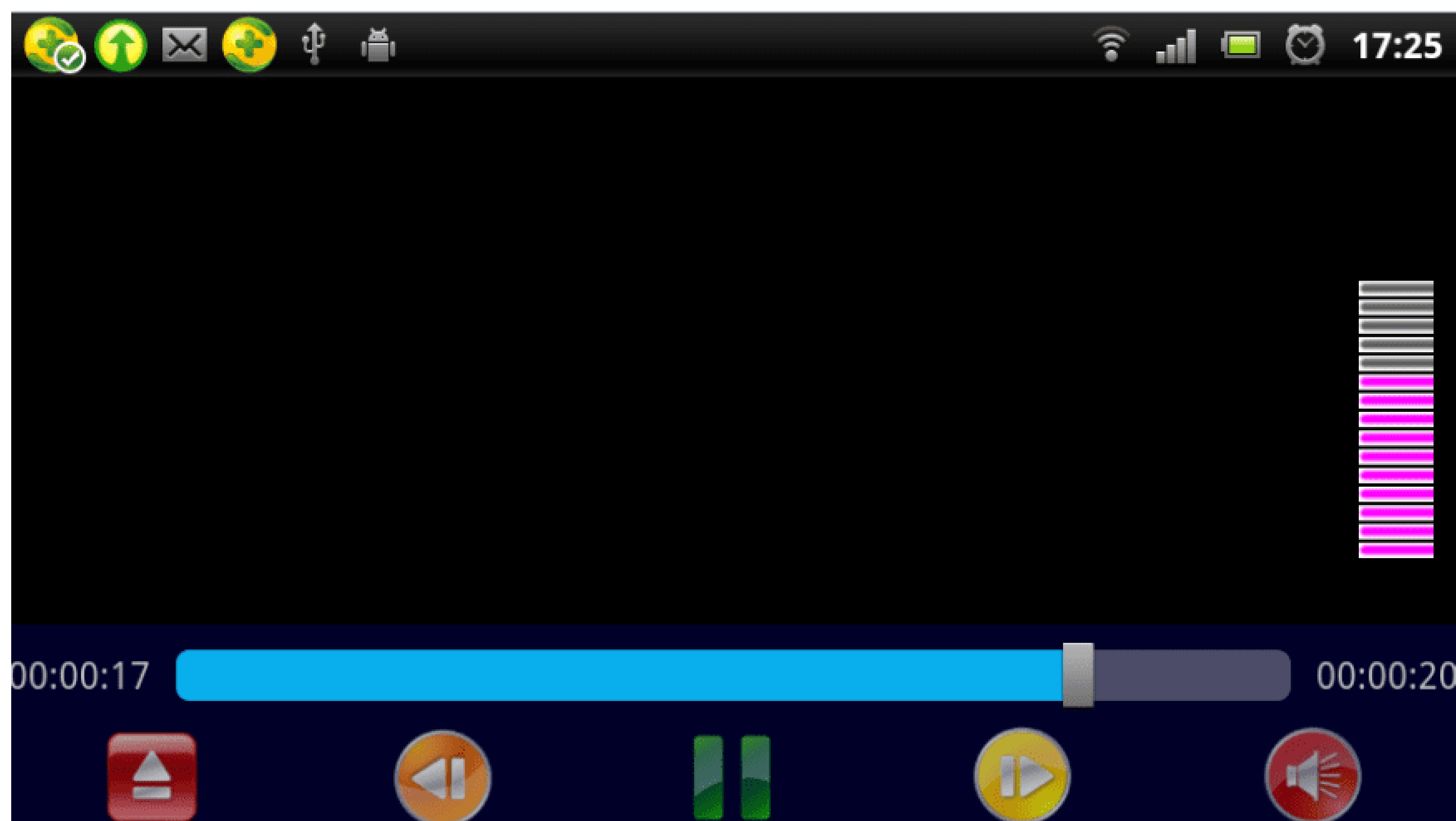


图 19.3　音量控制界面

以上就是我们视频播放器所需要设计的所有界面，接下来我们将结合播放流程对每个界面的代码实现和功能实现进行详细讲解。

19.2　播放器主界面

播放器主界面 main.xml 代码如下所示，可以看出整个界面只由一个我们自定义的视图类 com.guo.videoplayer.VideoView 组成。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
03      android:orientation="vertical"
04      android:layout_width="fill_parent"
05      android:layout_height="fill_parent"
06      android:gravity="center"
07      android:background="@color/background">
08      <!-- 播放主界面 -->
09      <com.guo.videoplayer.VideoView
10          android:id="@+id/vv"
11          android:layout_width="fill_parent"
12          android:layout_height="fill_parent"
13      />
14  </LinearLayout>
```

新建 VideoView.java 用于实现视频播放界面，代码如下所示。类 VideoView 继承于 SurfaceView 并实现 MediaPlayerControl 接口，在类的开始声明一些需要用到的变量，如视频的高宽、界面的高宽、媒体控制器、播放完成监听器、播放准备监听器等等。

播放视频时经常需要改变播放界面的尺寸，函数 setViewScale()用于设置视频窗口的大小。类 VideoView 有 3 个构造函数，对应了种不同的参数类型，分别为不带样式和属性、

带属性不带样式、带属性和样式，一般视图类都会实现这样 3 种构造函数。在构造函数中主要调用了 initVideoView，initVideoView 则为该 SurfaceView 对应的 SurfaceHolder 添加回调函数 mSHCallback()，如代码 073 行所示。

在回调函数中主要实现 3 个函数：onSurfaceCreated()、onSurfaceChanged() 和 onSurfaceDestroyed()，在 onSurfaceCreated()函数中调用 openVideo 打开视频文件，在 onSurfaceChanged()函数中播放打开的视频文件，在 onSurfaceDestroyed()函数中将使用的媒体播放器资源释放。

```
001 //视频播放界面
002 public class VideoView extends SurfaceView implementsMediaPlayer
    Control{
003     private String TAG = "VideoView";
004     //上下文
005     private Context mContext;
006     //视频路径和持续时间
007     private Uri         mUri;
008     private int         mDuration;
009
010     private SurfaceHolder mSurfaceHolder = null;
011     private MediaPlayer mMediaPlayer = null;
012     private boolean     mIsPrepared;
013     //视频的高宽
014     private int         mVideoWidth;
015     private int         mVideoHeight;
016     //播放界面的高宽
017     private int         mSurfaceWidth;
018     private int         mSurfaceHeight;
019     //媒体控制器
020     private MediaController mMediaController;
021     //播放完毕监听器
022     private OnCompletionListener mOnCompletionListener;
023     //播放准备监听器
024     private MediaPlayer.OnPreparedListener mOnPreparedListener;
025     private int         mCurrentBufferPercentage;
026     //出错监听器
027     private OnErrorListener mOnErrorListener;
028     private boolean     mStartWhenPrepared;
029     private int         mSeekWhenPrepared;
030     //尺寸改变监听器
031     private MySizeChangeLinstener mMyChangeLinstener;
032     //取得视频的宽
033     public int getVideoWidth(){
034         return mVideoWidth;
035     }
036     //取得视频的高
037     public int getVideoHeight(){
038         return mVideoHeight;
039     }
040     //设置视频播放窗口的高宽
041     public void setVideoScale(int width , int height){
042         LayoutParams lp = getLayoutParams();
043         lp.height = height;
044         lp.width = width;
045         setLayoutParams(lp);
046     }
047
```

```
048     //构造函数
049     public VideoView(Context context) {
050         super(context);
051         mContext = context;
052         //初始化视频界面
053         initVideoView();
054     }
055     //带属性构造函数
056     public VideoView(Context context, AttributeSet attrs) {
057         this(context, attrs, 0);
058         mContext = context;
059         //初始化视频界面
060         initVideoView();
061     }
062     //带属性、样式构造函数
063     public VideoView(Context context, AttributeSet attrs, int defStyle){
064         super(context, attrs, defStyle);
065         mContext = context;
066         //初始化视频界面
067         initVideoView();
068     }
069     //初始化界面
070     private void initVideoView() {
071         mVideoWidth = 0;
072         mVideoHeight = 0;
073         getHolder().addCallback(mSHCallback);
074         getHolder().setType(SurfaceHolder.SURFACE TYPE PUSH BUFFERS);
075         setFocusable(true);
076         setFocusableInTouchMode(true);
077         //请求焦点
078         requestFocus();
079     }
080     //surfaceview 回调函数
081     SurfaceHolder.Callback mSHCallback = new SurfaceHolder.Callback()
082     {
083         public void surfaceChanged(SurfaceHolder holder, int format,
084                                     int w, int h)
085         {
086             //取得播放界面的尺寸
087             mSurfaceWidth = w;
088             mSurfaceHeight = h;
089             if (mMediaPlayer != null && mIsPrepared && mVideoWidth == w
                 && mVideoHeight == h) {
090                 if (mSeekWhenPrepared != 0) {
091                     mMediaPlayer.seekTo(mSeekWhenPrepared);
092                     mSeekWhenPrepared = 0;
093                 }
094                 //开始播放视频
095                 mMediaPlayer.start();
096                 //并显示控制器界面
097                 if (mMediaController != null) {
098                     mMediaController.show();
099                 }
100             }
101         }
102         //打开视频
103         public void surfaceCreated(SurfaceHolder holder)
104         {
```

```
105             mSurfaceHolder = holder;
106             openVideo();
107         }
108         //界面销毁
109         public void surfaceDestroyed(SurfaceHolder holder)
110         {
111             //释放媒体播放器资源
112             mSurfaceHolder = null;
113             if (mMediaController != null) mMediaController.hide();
114             if (mMediaPlayer != null) {
115                 mMediaPlayer.reset();
116                 mMediaPlayer.release();
117                 mMediaPlayer = null;
118             }
119         }
120     };
```

打开视频的函数如下所示，该函数的主要功能是为视频播放做好准备工作。代码 008～010 行发送广播让已有的播放服务暂停，接下去释放已有的媒体播放器资源，并新建媒体播放器，设置准备监听器、完毕监听器、缓冲监听器、错误监听器等。

代码 082～096 行为尺寸改变监听器，当播放界面尺寸改变时，该函数将获取当前的视频尺寸，并对界面进行适配，同时执行回调函数 doMyThings()。代码 098～149 行为视频准备监听器，在该函数中首先适配视频的高宽，若检测到准备完成则播放视频并显示视频控制条。代码 150～186 行分别为播放完毕监听器、播放出错监听器、播放缓冲监听器，分别在视频播放完毕时、播放出错时、播放缓冲时调用。

```
001   //打开视频
002    private void openVideo() {
003        if (mUri == null || mSurfaceHolder == null) {
004            //当前未准备就绪
005            return;
006        }
007        //暂停
008        Intent i = new Intent("com.android.music.musicservicecommand");
009        i.putExtra("command", "pause");
010        mContext.sendBroadcast(i);
011        //释放资源
012        if (mMediaPlayer != null) {
013            mMediaPlayer.reset();
014            mMediaPlayer.release();
015            mMediaPlayer = null;
016        }
017        try {
018         //新建 meidiaplayer
019            mMediaPlayer = new MediaPlayer();
020            //设置准备监听器
021            mMediaPlayer.setOnPreparedListener(mPreparedListener);
022            //设置尺寸改变监听器
023            mMediaPlayer.setOnVideoSizeChangedListener(mSize
           ChangedListener);
024            mIsPrepared = false;
025            //重置视频播放时间
026            mDuration = -1;
027            //设置播放完毕监听器
028            mMediaPlayer.setOnCompletionListener(mCompletionListener);
029            //设置错误监听器
```

```
030            mMediaPlayer.setOnErrorListener(mErrorListener);
031            //设置缓冲更新监听器
032            mMediaPlayer.setOnBufferingUpdateListener(mBuffering
           UpdateListener);
033            mCurrentBufferPercentage = 0;
034            //设置视频文件路径
035            mMediaPlayer.setDataSource(mContext, mUri);
036            mMediaPlayer.setDisplay(mSurfaceHolder);
037            //设置音频流类型
038            mMediaPlayer.setAudioStreamType(AudioManager.STREAM_
           MUSIC);
039            //设置播放时屏幕一直点亮
040            mMediaPlayer.setScreenOnWhilePlaying(true);
041            //异步准备
042            mMediaPlayer.prepareAsync();
043            //绑定媒体控制器
044            attachMediaController();
045            //异常处理
046        } catch (IOException ex) {
047            Log.w(TAG, "Unable to open content: " + mUri, ex);
048            return;
049        } catch (IllegalArgumentException ex) {
050            Log.w(TAG, "Unable to open content: " + mUri, ex);
051            return;
052        }
053    }
054    //设置媒体播放器控制器
055    public void setMediaController(MediaController controller) {
056        if (mMediaController != null) {
057            mMediaController.hide();
058        }
059        mMediaController = controller;
060      //绑定媒体控制器
061        attachMediaController();
062    }
063    //绑定媒体控制器
064    private void attachMediaController() {
065        if (mMediaPlayer != null && mMediaController != null) {
066            mMediaController.setMediaPlayer(this);
067            View anchorView = this.getParent() instanceof View ?
068                    (View)this.getParent() : this;
069            mMediaController.setAnchorView(anchorView);
070            mMediaController.setEnabled(mIsPrepared);
071        }
072    }
073    //自定义回调函数接口
074    public interface MySizeChangeLinstener{
075     public void doMyThings();
076    }
077    //取得尺寸改变监听器
078    public void setMySizeChangeLinstener(MySizeChangeLinstener l){
079     mMyChangeLinstener = l;
080    }
081    //尺寸改变监听器
082    MediaPlayer.OnVideoSizeChangedListener mSizeChangedListener =
083        new MediaPlayer.OnVideoSizeChangedListener() {
084            public void onVideoSizeChanged(MediaPlayer mp, int width, int
            height) {
085             //取得当前的尺寸
```

```
086                mVideoWidth = mp.getVideoWidth();
087                mVideoHeight = mp.getVideoHeight();
088                if(mMyChangeLinstener!=null){
089                 mMyChangeLinstener.doMyThings();
090                }
091                //设置尺寸
092                if (mVideoWidth != 0 && mVideoHeight != 0) {
093                    getHolder().setFixedSize(mVideoWidth, mVideoHeight);
094                }
095            }
096    };
097    //媒体播放器准备监听器
098    MediaPlayer.OnPreparedListener mPreparedListener = new
        MediaPlayer.OnPreparedListener() {
099        public void onPrepared(MediaPlayer mp) {
100            mIsPrepared = true;
101            if (mOnPreparedListener != null) {
102                mOnPreparedListener.onPrepared(mMediaPlayer);
103            }
104            //设置该视图的可用状态
105            if (mMediaController != null) {
106                mMediaController.setEnabled(true);
107            }
108            //取得视频文件的宽高信息
109            mVideoWidth = mp.getVideoWidth();
110            mVideoHeight = mp.getVideoHeight();
111            if (mVideoWidth != 0 && mVideoHeight != 0) {
112                //为界面设置高宽信息
113                getHolder().setFixedSize(mVideoWidth, mVideoHeight);
114                if (mSurfaceWidth == mVideoWidth && mSurfaceHeight ==
                    mVideoHeight) {
115                    //设置播放初始位置
116                    if (mSeekWhenPrepared != 0) {
117                        mMediaPlayer.seekTo(mSeekWhenPrepared);
118                        mSeekWhenPrepared = 0;
119                    }
120                    //当准备完毕，播放视频
121                    if (mStartWhenPrepared) {
122                        mMediaPlayer.start();
123                        mStartWhenPrepared = false;
124                        //显示控制器
125                        if (mMediaController != null) {
126                            mMediaController.show();
127                        }
128                    } else if (!isPlaying() &&
129                            (mSeekWhenPrepared != 0 || getCurrentPosition()
                            > 0)) {
130                       if (mMediaController != null) {
131                           //当暂停时显示控制器
132                           mMediaController.show(0);
133                       }
134                   }
135                }
136            } else {
137                //未知视频尺寸时仍然播放该视频
138                if (mSeekWhenPrepared != 0) {
139                    mMediaPlayer.seekTo(mSeekWhenPrepared);
140                    mSeekWhenPrepared = 0;
141                }
```

```
142                //开始播放
143                if (mStartWhenPrepared) {
144                    mMediaPlayer.start();
145                    mStartWhenPrepared = false;
146                }
147            }
148        }
149    };
150    //播放完毕时调用
151    private MediaPlayer.OnCompletionListener mCompletionListener =
152        new MediaPlayer.OnCompletionListener() {
153        public void onCompletion(MediaPlayer mp) {
154         //隐藏控制器
155            if (mMediaController != null) {
156                mMediaController.hide();
157            }
158            if (mOnCompletionListener != null) {
159                mOnCompletionListener.onCompletion(mMediaPlayer);
160            }
161        }
162    };
163    //错误监听器
164    private MediaPlayer.OnErrorListener mErrorListener =
165        new MediaPlayer.OnErrorListener() {
166        public boolean onError(MediaPlayer mp, int framework err, int
       impl err) {
167            //隐藏控制器
168            if (mMediaController != null) {
169                mMediaController.hide();
170            }
171            if (mOnErrorListener != null) {
172                if (mOnErrorListener.onError(mMediaPlayer, framework err
               , impl err)) {
173                    return true;
174                }
175            }
176            return true;
177        }
178    };
179    //缓冲监听器
180    private MediaPlayer.OnBufferingUpdateListener mBufferingUpdate
   Listener =
181        new MediaPlayer.OnBufferingUpdateListener() {
182        public void onBufferingUpdate(MediaPlayer mp, int percent) {
183         //设置当前缓冲百分比
184            mCurrentBufferPercentage = percent;
185        }
186    };
187    //设置准备监听器
188    public void setOnPreparedListener(MediaPlayer.OnPreparedListener l)
189    {
190        mOnPreparedListener = l;
191    }
192    //设置播放完毕监听器
193    public void setOnCompletionListener(OnCompletionListener l)
194    {
195        mOnCompletionListener = l;
196    }
197    //设置出错监听器
198    public void setOnErrorListener(OnErrorListener l)
```

```
199    {
200        mOnErrorListener = l;
201    }
```

除了打开视频函数外，作为视频播放类，还需要实现视频播放的流程控制函数。具体代码如下所示，代码 001～029 行用于设置视频播放的尺寸，030～044 行用于设置视频文件的路径。函数 stopPlayback、start、pause、seekTo 分别用于停止播放、开始播放、暂停播放、跳转到指定位置播放。onTouchEvent 用于响应界面触摸事件，当单击视频界面时，视频控制条将在显示和隐藏之间切换。

此外，还实现了一些获取播放状态的相关函数，如 getDuration()函数用于获取视频的持续时间，getCurrentPosition 用于获取当前播放的位置，函数 isPlaying()用于获取当前的播放状态（播放或暂停），函数 getBufferPercentage 用于获取当前缓冲的百分比。

```
001   @Override
002    protected void onMeasure(int widthMeasureSpec,intheightMeasureSpec)
       {
003        //传递尺寸信息
004        int width = getDefaultSize(mVideoWidth, widthMeasureSpec);
005        int height = getDefaultSize(mVideoHeight, heightMeasureSpec);
006        setMeasuredDimension(width,height);
007    }
008    //调整界面尺寸适应分辨率
009    public int resolveAdjustedSize(int desiredSize, int measureSpec) {
010        int result = desiredSize;
011        int specMode = MeasureSpec.getMode(measureSpec);
012        int specSize =  MeasureSpec.getSize(measureSpec);
013
014        switch (specMode) {
015        //无限制
016        case MeasureSpec.UNSPECIFIED:
017            result = desiredSize;
018            break;
019        //不能超过限制尺寸
020        case MeasureSpec.AT MOST:
021            result = Math.min(desiredSize, specSize);
022            break;
023        //精确设置尺寸
024        case MeasureSpec.EXACTLY:
025            result = specSize;
026            break;
027        }
028        return result;
029}
030    //设置视频路径
031    public void setVideoPath(String path) {
032        setVideoURI(Uri.parse(path));
033    }
034    //设置视频 uri 地址
035    public void setVideoURI(Uri uri) {
036        mUri = uri;
037        mStartWhenPrepared = false;
038        mSeekWhenPrepared = 0;
039        //打开视频
040        openVideo();
041        requestLayout();
042        //更新界面
```

```
043         invalidate();
044     }
045     //停止播放
046     public void stopPlayback() {
047         if (mMediaPlayer != null) {
048             mMediaPlayer.stop();
049             mMediaPlayer.release();
050             mMediaPlayer = null;
051         }
052     }
053     //响应触摸事件、暂停/播放切换
054     @Override
055     public boolean onTouchEvent(MotionEvent ev) {
056        if (mIsPrepared && mMediaPlayer != null && mMediaController != null)
           {
057             toggleMediaControlsVisiblity();
058         }
059         return false;
060     }
061     //视频播放控制器隐藏、显示切换
062     private void toggleMediaControlsVisiblity() {
063         if (mMediaController.isShowing()) {
064             mMediaController.hide();
065         } else {
066             mMediaController.show();
067         }
068     }
069     //开始播放
070     public void start() {
071         if (mMediaPlayer != null && mIsPrepared) {
072                 mMediaPlayer.start();
073                 mStartWhenPrepared = false;
074         } else {
075             mStartWhenPrepared = true;
076         }
077     }
078     //暂停播放
079     public void pause() {
080         if (mMediaPlayer != null && mIsPrepared) {
081             if (mMediaPlayer.isPlaying()) {
082                 mMediaPlayer.pause();
083             }
084         }
085         mStartWhenPrepared = false;
086     }
087     //取得视频的持续时间
088     public int getDuration() {
089         if (mMediaPlayer != null && mIsPrepared) {
090             if (mDuration > 0) {
091                 return mDuration;
092             }
093             //获得播放时间
094             mDuration = mMediaPlayer.getDuration();
095             return mDuration;
096         }
097         mDuration = -1;
098         return mDuration;
099     }
100     //取得当前播放的位置
101     public int getCurrentPosition() {
```

```
102         if (mMediaPlayer != null && mIsPrepared) {
103             return mMediaPlayer.getCurrentPosition();
104         }
105         return 0;
106     }
107     //跳转到指定进度
108     public void seekTo(int msec) {
109         if (mMediaPlayer != null && mIsPrepared) {
110             mMediaPlayer.seekTo(msec);
111         } else {
112             mSeekWhenPrepared = msec;
113         }
114     }
115     //返回播放器播放的状态
116     public boolean isPlaying() {
117         if (mMediaPlayer != null && mIsPrepared) {
118             return mMediaPlayer.isPlaying();
119         }
120         return false;
121     }
122 
123     public int getBufferPercentage() {
124         if (mMediaPlayer != null) {
125             return mCurrentBufferPercentage;
126         }
127         return 0;
128     }
```

19.3　播放器功能实现

（1）整个播放器的所有功能均在 VideoPlayerActivity 中实现，代码如下所示。在类的开始声明需要用到的变量，playList 用于存放视频播放列表信息。类 MovieInfo 用于存放每个视频的信息，包括视频名称和文件路径。videoListUri 为视频文件的数据库查询地址，通过这个地址可以查询到记录在系统媒体库的视频信息。其他变量都是和视频播放界面相关的，如播放时间、播放进度条、按键等。

```
01  //播放视频主界面功能实现
02  public class VideoPlayerActivity extends Activity {
03      //播放列表
04      public static LinkedList<MovieInfo> playList = newLinkedList
        <MovieInfo>();
05      //视频信息类
06      public class MovieInfo{
07          //电影名称
08          String displayName;
09          //文件路径
10          String path;
11      }
12      //媒体文件数据库查询地址
13      private Uri videoListUri =MediaStore.Video.Media.EXTERNAL_
        CONTENT_URI;
14      //播放进度条位置
15      private static int position ;
```

```
16      //播放时间
17      private int playedTime;
18      //视频视图界面
19      private VideoView vv = null;
20      //进度条
21      private SeekBar seekBar = null;
22      //视频总时间
23      private TextView durationTextView = null;
24      //已播放时间
25      private TextView playedTextView = null;
26      //手势检测器
27      private GestureDetector mGestureDetector = null;
28      //声音管理
29      private AudioManager mAudioManager = null;
30      //最大音量、当前音量
31      private int maxVolume = 0;
32      private int currentVolume = 0;
33      //视频列表
34      private ImageButton bn1 = null;
35      //前一个视频
36      private ImageButton bn2 = null;
37      //播放/暂停
38      private ImageButton bn3 = null;
39      //后一个视频
40      private ImageButton bn4 = null;
41      //音量控制
42      private ImageButton bn5 = null;
43      //控制视图
44      private View controlView = null;
45      //弹出窗口
46      private PopupWindow controler = null;
47      //声音控制视图
48      private SoundView mSoundView = null;
49      private PopupWindow mSoundWindow = null;
50      //屏幕高宽
51      private static int screenWidth = 0;
52      private static int screenHeight = 0;
53      private static int controlHeight = 0;
54
55      private final static int TIME = 6868;
56      //标志位
57      private boolean isControllerShow = true;
58      private boolean isPaused = false;
59      private boolean isFullScreen = false;
60      private boolean isSilent = false;
61      private boolean isSoundShow = false;
```

（2）当打开视频播放器时，首先调用 onCreate()函数，代码如下所示。代码 006～019 行添加 idle 处理器，即程序处理完所有的消息后将执行该处理器的 queueIdle()函数，在该函数中调用 controler.update 更新控制条的位置。接下去膨胀出控制条界面 controler 和音量控制界面，初始化控制条界面的所有相关元素并为音量控制界面设置音量改变监听器。当系统音量改变时，将调用 setYourVolume 将音量信息更新到音量界面。

代码 052～057 行获取通过 intent 传递过来的视频信息，并传递给视频播放界面 vv。代码 063 行通过函数 getVideoFile()获取路径/sdcard/下的所有满足条件的视频文件，并将视

频文件保存到 playList 中。为了全面起见，我们通过查询系统媒体库获得另一个视频播放列表 playList2，然后取 playList 和 playList2 中元素较多者作为当前的播放列表。

```
001   @Override
002   public void onCreate(Bundle savedInstanceState) {
003       super.onCreate(savedInstanceState);
004       setContentView(R.layout.main);
005       //添加 idle 处理器
006       Looper.myQueue().addIdleHandler(new IdleHandler(){
007        //当当前消息队列中的所有体验消息都执行完调用
008        @Override
009        public boolean queueIdle() {
010            //显示控制条
011            if(controler != null && vv.isShown()){
012                controler.showAtLocation(vv, Gravity.BOTTOM, 0, 0);
013                //更新控制条的位置
014                controler.update(0, 0, screenWidth, controlHeight);
015            }
016            //返回 false 将使该 handler 调用一次之后被移除
017            return false;
018        }
019       });
020       //膨胀出控制条
021       controlView = getLayoutInflater().inflate(R.layout.controler,
          null);
022       controler = new PopupWindow(controlView);
023       //视频持续时间
024       durationTextView = (TextView) controlView.findViewById
          (R.id.duration);
025       //已播放时间
026       playedTextView = (TextView) controlView.findViewById(R.id.has_
          played);
027       //音量控制界面
028       mSoundView = new SoundView(this);
029       mSoundView.setOnVolumeChangeListener(new
          OnVolumeChangedListener(){
030        //设置音量
031        @Override
032        public void setYourVolume(int index) {
033            //移除消息队列的消息
034            cancelDelayHide();
035            //更新音量大小
036            updateVolume(index);
037            //延迟隐藏控制器
038            hideControllerDelay();
039        }
040       });
041       //获得音量控制界面
042       mSoundWindow = new PopupWindow(mSoundView);
043       position = -1;
044       //初始化 5 个按钮
045       bn1 = (ImageButton) controlView.findViewById(R.id.button1);
046       bn2 = (ImageButton) controlView.findViewById(R.id.button2);
047       bn3 = (ImageButton) controlView.findViewById(R.id.button3);
048       bn4 = (ImageButton) controlView.findViewById(R.id.button4);
049       bn5 = (ImageButton) controlView.findViewById(R.id.button5);
050       //初始化视频播放界面
051       vv = (VideoView) findViewById(R.id.vv);
```

```
052         //取得视频路径
053         Uri uri = getIntent().getData();
054         if(uri!=null){
055          if(vv.getVideoHeight()==0){
056              vv.setVideoURI(uri);
057          }
058          bn3.setImageResource(R.drawable.pause);
059         }else{
060          bn3.setImageResource(R.drawable.play);
061         }
062         //取得指定路径下的所有视频列表
063         getVideoFile(playList, new File("/sdcard/"));
064         //取得媒体库中的视频信息
065         Cursor cursor = getContentResolver().query(videoListUri, new
             String[]{"_display_name","_data"}, null, null, null);
066         int n = cursor.getCount();
067         cursor.moveToFirst();
068         LinkedList<MovieInfo> playList2 = new LinkedList<MovieInfo>();
069         //遍历得到的数据，将媒体信息保存到 playList2 中
070         for(int i = 0 ; i != n ; ++i){
071          MovieInfo mInfo = new MovieInfo();
072          mInfo.displayName = cursor.getString(cursor.getColumnIndex
             ("_display_name"));
073          mInfo.path = cursor.getString(cursor.getColumnIndex("_data"));
074          playList2.add(mInfo);
075          cursor.moveToNext();
076         }
077         //比较这两种方式获得的视频数目，取较大者
078         if(playList2.size() > playList.size()){
079          playList = playList2;
080         }
081         //当播放视频窗口大小改变时
082         vv.setMySizeChangeLinstener(new MySizeChangeLinstener(){
083          @Override
084          public void doMyThings() {
085              //设置视频播放窗口的高宽
086              setVideoScale(SCREEN_DEFAULT);
087          }
088 
089         });
090         //设置按键的透明度
091         bn1.setAlpha(0xBB);
092         bn2.setAlpha(0xBB);
093         bn3.setAlpha(0xBB);
094         bn4.setAlpha(0xBB);
095         //取得声音管理器
096         mAudioManager = (AudioManager) getSystemService(Context.AUDIO_
             SERVICE);
097         maxVolume = mAudioManager.getStreamMaxVolume(AudioManager.
            STREAM_MUSIC);
098         currentVolume = mAudioManager.getStreamVolume(
             AudioManager.STREAM_MUSIC);
099         //根据当前音量大小设置按键的透明度
100         bn5.setAlpha(findAlphaFromSound());
```

（3）获取视频列表的函数实现方式如下所示，递归搜索指定的目录，并将文件扩展名为 mp4 和 3gp 的文件加入到播放列表中。

```
01  private void getVideoFile(final LinkedList<MovieInfo> list,File file){
```

```
02      //取得满足条件的视频文件
03      file.listFiles(new FileFilter(){
04          @Override
05          public boolean accept(File file) {
06              // TODO Auto-generated method stub
07              String name = file.getName();
08              int i = name.indexOf('.');
09              if(i != -1){
10                  name = name.substring(i);
11                  //若文件的扩展名为 mp4 或者 3gp
12                  if(name.equalsIgnoreCase(".mp4")||name.equalsIgnore
                   Case(".3gp")){
13                      //将符合条件的视频文件信息保存到 list 中
14                      MovieInfo mi = new MovieInfo();
15                      //获得文件名
16                      mi.displayName = file.getName();
17                      //获得视频文件的路径
18                      mi.path = file.getAbsolutePath();
19                      list.add(mi);
20                      return true;
21                  }
22              }else if(file.isDirectory()){
23                  getVideoFile(list, file);
24              }
25              return false;
26          }
27      });
28    }
```

（4）在 onCreate()函数最后还要实现相关的按键监听函数，代码 001～012 行为“弹出”按钮绑定监听器，可以看出单击该按钮将弹出 VideoChooseActivity，用于选择视频。后面我们会再讲到这个视频选择界面。

btn4、btn3、btn2 分别实现播放上一个视频、播放/暂停切换、播放下一个视频的功能。btn5 为音量键，设有单击和长按监听器，单击时将显示和隐藏音量增减界面，长按时将切换静音和取消静音。当拖动进度条时，将调用 VideoView 的 seekTo()函数从指定位置开始播放视频。除了按键监听器外，这里还设置了手势检测器，单击视频界面时将显示或隐藏控制条，双击视频界面时将在默认方式和全屏方式之间切换显示，长按时将暂停或者播放视频。最后，为视频界面设置准备监听器和播放完毕监听器，在准备监听器中设置视频的播放尺寸并显示使视频的持续时间到指定位置；在播放完毕监听器中，设置自动播放下一个视频，即类似顺序播放的功能。

```
001 //打开视频播放列表按钮
002 bn1.setOnClickListener(new OnClickListener(){
003     @Override
004     public void onClick(View arg0) {
005         // TODO Auto-generated method stub
006         Intent intent = new Intent();
007         intent.setClass(VideoPlayerActivity.this,
             VideoChooseActivity.class);
008         //显示视频播放列表
009         VideoPlayerActivity.this.startActivityForResult(intent, 0);
010         cancelDelayHide();
011     }
012 });
013 //播放前一个视频
014 bn4.setOnClickListener(new OnClickListener(){
```

```
015     @Override
016     public void onClick(View v) {
017         //取得前一个视频并播放
018         int n = playList.size();
019         if(++position < n){
020             vv.setVideoPath(playList.get(position).path);
021             cancelDelayHide();
022             hideControllerDelay();
023         }else{
024             VideoPlayerActivity.this.finish();
025         }
026     }
027
028 });
029 //播放、暂停切换按钮
030 bn3.setOnClickListener(new OnClickListener(){
031     @Override
032     public void onClick(View v) {
033         // TODO Auto-generated method stub
034         cancelDelayHide();
035         //当前暂停，则变为播放
036         if(isPaused){
037             vv.start();
038             bn3.setImageResource(R.drawable.pause);
039             hideControllerDelay();
040         }else{
041             vv.pause();
042             bn3.setImageResource(R.drawable.play);
043         }
044         //改变暂停播放标志位
045         isPaused = !isPaused;
046     }
047 });
048 //下一个视频
049 bn2.setOnClickListener(new OnClickListener(){
050     @Override
051     public void onClick(View v) {
052         //播放下一个视频
053         if(--position>=0){
054             vv.setVideoPath(playList.get(position).path);
055             cancelDelayHide();
056             hideControllerDelay();
057         }else{
058             VideoPlayerActivity.this.finish();
059         }
060     }
061 });
062 //显示音量控制界面
063 bn5.setOnClickListener(new OnClickListener(){
064 @Override
065 public void onClick(View v) {
066     // TODO Auto-generated method stub
067     cancelDelayHide();
068     //如果已经显示则隐藏
069     if(isSoundShow){
070         mSoundWindow.dismiss();
071     }else{
072         //显示音量控制界面
073         if(mSoundWindow.isShowing()){
074             mSoundWindow.update(15,0,SoundView.MY_WIDTH,SoundView
```

```
              MY HEIGHT);
075          }else{
076              mSoundWindow.showAtLocation(vv, Gravity.RIGHT|Gravity.
                CENTER VERTICAL, 15, 0);
077              mSoundWindow.update(15,0,SoundView.MY WIDTH,SoundView.MY
                HEIGHT);
078          }
079      }
080      isSoundShow = !isSoundShow;
081      hideControllerDelay();
082 }
083   });
084 //设置音量键长按监听器
085 bn5.setOnLongClickListener(new OnLongClickListener(){
086     @Override
087     public boolean onLongClick(View arg0) {
088         //静音、非静音切换
089         if(isSilent){
090             bn5.setImageResource(R.drawable.soundenable);
091         }else{
092             bn5.setImageResource(R.drawable.sounddisable);
093         }
094         isSilent = !isSilent;
095         //更新音量
096         updateVolume(currentVolume);
097         //去除隐藏消息
098         cancelDelayHide();
099         //发送隐藏消息
100         hideControllerDelay();
101         return true;
102     }
103 });
104 //进度条
105 seekBar = (SeekBar) controlView.findViewById(R.id.seekbar);
106 //进度条进度改变监听器
107 seekBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener(){
108         @Override
109         public void onProgressChanged(SeekBar seekbar, int progress,
           boolean fromUser) {
110             //改变到指定位置
111             if(fromUser){
112                 vv.seekTo(progress);
113             }
114 
115         }
116         //按下的时候触发
117         @Override
118         public void onStartTrackingTouch(SeekBar arg0) {
119             myHandler.removeMessages(HIDE CONTROLER);
120         }
121         //离开进度条的时候触发
122         @Override
123         public void onStopTrackingTouch(SeekBar seekBar) {
124             // TODO Auto-generated method stub
125             myHandler.sendEmptyMessageDelayed(HIDE_CONTROLER, TIME);
126         }
127     });
128 //取得屏幕大小
129 getScreenSize();
```

```
130  //手势检测器
131 mGestureDetector = new GestureDetector(new SimpleOnGestureListener(){
132     //双击
133     @Override
134     public boolean onDoubleTap(MotionEvent e) {
135         //进行视频播放窗口大小切换
136         if(isFullScreen){
137             setVideoScale(SCREEN_DEFAULT);
138         }else{
139             setVideoScale(SCREEN_FULL);
140         }
141         //全屏显示标志位
142         isFullScreen = !isFullScreen;
143         //显示控制器界面
144         if(isControllerShow){
145             showController();
146         }
147         return true;
148     }
149     //单击
150     @Override
151     public boolean onSingleTapConfirmed(MotionEvent e) {
152         //显示控制器界面
153         if(!isControllerShow){
154             showController();
155             hideControllerDelay();
156         }else {
157             //隐藏控制器界面
158             cancelDelayHide();
159             hideController();
160         }
161         return true;
162     }
163     //长按
164     @Override
165     public void onLongPress(MotionEvent e) {
166         //播放、暂停切换
167         if(isPaused){
168             vv.start();
169             bn3.setImageResource(R.drawable.pause);
170             cancelDelayHide();
171             hideControllerDelay();
172         }else{
173             vv.pause();
174             bn3.setImageResource(R.drawable.play);
175             cancelDelayHide();
176             showController();
177         }
178         //更改暂停标志位
179         isPaused = !isPaused;
180     }
181 });
182 //当媒体文件载入时调用
183 vv.setOnPreparedListener(new OnPreparedListener(){
184         @Override
185         public void onPrepared(MediaPlayer arg0) {
186             //设置视频播放尺寸
187             setVideoScale(SCREEN_DEFAULT);
188             //默认尺寸播放
```

```
189             isFullScreen = false;
190             if(isControllerShow){
191                 showController();
192             }
193             //获得视频持续时间
194             int i = vv.getDuration();
195             seekBar.setMax(i);
196             i/=1000;
197             int minute = i/60;
198             int hour = minute/60;
199             int second = i%60;
200             minute %= 60;
201             //显示视频持续时间
202             durationTextView.setText(String.format("%02d:%02d:%02d",
                hour,minute,second));
203             vv.start();
204             bn3.setImageResource(R.drawable.pause);
205             hideControllerDelay();
206             myHandler.sendEmptyMessage(PROGRESS CHANGED);
207         }
208     });
209 //视频播放完毕调用
210 vv.setOnCompletionListener(new OnCompletionListener(){
211         @Override
212         public void onCompletion(MediaPlayer arg0) {
213             //取得播放列表大小
214             int n = playList.size();
215             //循环播放
216             if(++position < n){
217                 vv.setVideoPath(playList.get(position).path);
218             }else{
219                 VideoPlayerActivity.this.finish();
220             }
221         }
222     });
```

（5）上面用到的一些函数的实现代码如下所示。onActivityResult()函数用于将播放列表界面传递过来的视频路径信息设置到 vv 中，从而播放选中的视频。myHandler 用来处理进度条改变事件和隐藏控制条事件，setVideoScale()函数用于根据传入的参数设置视频播放尺寸为默认大小或者全屏，findAlphaFromSound 则根据当前的音量大小返回一个透明度的值，updateVolume()函数用于设置当前音量。

```
001   @Override
002 protected void onActivityResult(int requestCode, int resultCode, Intent
    data) {
003     // TODO Auto-generated method stub
004     if(requestCode==0&&resultCode==Activity.RESULT OK){
005         int result = data.getIntExtra("CHOOSE", -1);
006         if(result!=-1){
007             //播放指定视频
008             vv.setVideoPath(playList.get(result).path);
009             position = result;
010         }
011         return ;
012     }
013     super.onActivityResult(requestCode, resultCode, data);
014 }
015   //改变播放进度条事件 ID
016 private final static int PROGRESS_CHANGED = 0;
```

```
017 //隐藏控制器事件 ID
018   private final static int HIDE_CONTROLER = 1;
019   //消息处理器
020   Handler myHandler = new Handler(){
021    @Override
022    public void handleMessage(Message msg) {
023        // TODO Auto-generated method stub
024
025        switch(msg.what){
026            //进度条改变
027            case PROGRESS_CHANGED:
028                //进度条移动到指定位置
029                int i = vv.getCurrentPosition();
030                seekBar.setProgress(i);
031                //显示已播放时间
032                i/=1000;
033                int minute = i/60;
034                int hour = minute/60;
035                int second = i%60;
036                minute %= 60;
037                playedTextView.setText(String.format("%02d:%02d:%02d",
                   hour,minute,second));
038                sendEmptyMessage(PROGRESS_CHANGED);
039                break;
040            case HIDE_CONTROLER:
041                //隐藏控制条
042                hideController();
043                break;
044        }
045        super.handleMessage(msg);
046    }
047   };
048 //取得屏幕的尺寸
049 private void getScreenSize()
050 {
051    Display display = getWindowManager().getDefaultDisplay();
052    screenHeight = display.getHeight();
053    screenWidth = display.getWidth();
054    controlHeight = screenHeight/4;
055 }
056 //隐藏控制界面
057 private void hideController(){
058    //隐藏视频控制条
059    if(controler.isShowing()){
060        controler.update(0,0,0, 0);
061        isControllerShow = false;
062    }
063    //隐藏音量控制界面
064    if(mSoundWindow.isShowing()){
065        mSoundWindow.dismiss();
066        isSoundShow = false;
067    }
068 }
069 //延迟发送隐藏控制界面的消息
070 private void hideControllerDelay(){
071    myHandler.sendEmptyMessageDelayed(HIDE_CONTROLER, TIME);
072 }
073 //显示控制界面
074 private void showController(){
```

```
075     controler.update(0,0,screenWidth, controlHeight);
076     isControllerShow = true;
077 }
078 //移除消息队列中待处理隐藏控制条的消息
079 private void cancelDelayHide(){
080     myHandler.removeMessages(HIDE_CONTROLER);
081 }
082 //屏幕状态标志
083   private final static int SCREEN_FULL = 0;
084   private final static int SCREEN_DEFAULT = 1;
085   //设置显示比例
086   private void setVideoScale(int flag){
087     vv.getLayoutParams();
088     switch(flag){
089     //全屏显示
090     case SCREEN_FULL:
091         vv.setVideoScale(screenWidth, screenHeight);
092         getWindow().addFlags(WindowManager.LayoutParams.
        FLAG_FULLSCREEN);
093
094         break;
095     //默认显示
096     case SCREEN_DEFAULT:
097         //获得当前播放窗口的高宽
098         int videoWidth = vv.getVideoWidth();
099         int videoHeight = vv.getVideoHeight();
100         //获得当前屏幕的高宽
101         int mWidth = screenWidth;
102         int mHeight = screenHeight - 25;
103         //适配比例
104         if (videoWidth > 0 && videoHeight > 0) {
105             if ( videoWidth * mHeight  > mWidth * videoHeight ) {
106                 mHeight = mWidth * videoHeight / videoWidth;
107             } else if ( videoWidth * mHeight  < mWidth * videoHeight ) {
108                 mWidth = mHeight * videoWidth / videoHeight;
109             }
110         }
111         //设置视频显示的尺寸
112         vv.setVideoScale(mWidth, mHeight);
113
114         getWindow().clearFlags(WindowManager.LayoutParams.FLAG_
        FULLSCREEN);
115
116         break;
117     }
118   }
119   //根据当前的音量值决定按钮显示的透明度
120   private int findAlphaFromSound(){
121     if(mAudioManager!=null){
122         int alpha = currentVolume * (0xCC-0x55) / maxVolume + 0x55;
123         return alpha;
124     }else{
125         return 0xCC;
126     }
127   }
128   //更新音量
129   private void updateVolume(int index){
130     if(mAudioManager!=null){
131         if(isSilent){
```

```
132             //静音
133             mAudioManager.setStreamVolume(AudioManager.STREAM_MUSIC, 0
                , 0);
134         }else{
135             mAudioManager.setStreamVolume(AudioManager.STREAM_MUSIC,
                 index, 0);
136         }
137         currentVolume = index;
138         //设置音量键的透明度
139         bn5.setAlpha(findAlphaFromSound());
140     }
141    }
```

（6）为了使手势控制有效，需要在 onTouchEvent 中将手势信息传递给 mGestureDetector，onConfigurationChanged() 函数将响应 AndroidManifest.xml 中的 android:configChanges 指定的事件，这里我们指定 android:configChanges="keyboardHidden|orientation"，即键盘隐藏和方向改变都将触发这个函数。

最后实现 Activity 的一些生命周期函数 onPause()、onResume()和 onDestroy()。在 onPause()中暂停视频并保存当前的播放位置，在 onResume()中获取视频的播放位置并开始播放，在 onDestroy()中关闭当前显示的 mSoundWindow 并释放其他相关资源。

```
01  @Override
02  public boolean onTouchEvent(MotionEvent event) {
03      // TODO Auto-generated method stub
04      //触发单击事件
05      boolean result = mGestureDetector.onTouchEvent(event);
06      return result;
07  }
08  //响应 android:configChanges 中指定的事件
09  @Override
10  public void onConfigurationChanged(Configuration newConfig) {
11      // TODO Auto-generated method stub
12      //重新获得屏幕尺寸
13      getScreenSize();
14      if(isControllerShow){
15          //取消延迟隐藏
16          cancelDelayHide();
17          //隐藏控制器
18          hideController();
19          //显示控制器
20          showController();
21          //发送延迟隐藏的消息
22          hideControllerDelay();
23      }
24      super.onConfigurationChanged(newConfig);
25  }
26  @Override
27  protected void onPause() {
28      //保存当前的播放时间
29      playedTime = vv.getCurrentPosition();
30      //暂停播放
31      vv.pause();
32      //改变按钮
33      bn3.setImageResource(R.drawable.play);
34      super.onPause();
35  }
```

```
36  @Override
37  protected void onResume() {
38      //恢复到之前播放的位置开始播放
39      vv.seekTo(playedTime);
40      vv.start();
41      if(vv.getVideoHeight()!=0){
42          bn3.setImageResource(R.drawable.pause);
43          hideControllerDelay();
44      }
45      super.onResume();
46  }
47  @Override
48  protected void onDestroy() {
49      // TODO Auto-generated method stub
50      //关闭控制器
51      if(controler.isShowing()){
52          controler.dismiss();
53      }
54      //关闭音量控制界面
55      if(mSoundWindow.isShowing()){
56          mSoundWindow.dismiss();
57      }
58      //移除消息队列中的消息
59      myHandler.removeMessages(PROGRESS_CHANGED);
60      myHandler.removeMessages(HIDE_CONTROLER);
61      //移除播放列表
62      playList.clear();
63      super.onDestroy();
64  }
```

（7）音量控制界面的实现类如下所示。在构造函数中调用 init()函数进行初始化，包括获取图片资源和获取系统当前音量值，并将值进行比例转化存储到 index，在 onDraw()中将音量绘制出来。绘制方法是根据当前 index 的值，从下到上绘制 index 个彩色横杠和 15-index 个灰色横杠。在 onTouchEvent()中根据触摸的 Y 坐标按比例转化成音量的值 index，并重新绘制图像。

```
01  //音量控制界面
02  public class SoundView extends View{
03      public final static String TAG = "SoundView";
04      //上下文
05      private Context mContext;
06      private Bitmap bm , bm1;
07      private int bitmapWidth , bitmapHeight;
08      //音量大小
09      private int index;
10      //音量改变监听器
11      private OnVolumeChangedListener mOnVolumeChangedListener;
12      private final static int HEIGHT = 11;
13      public final static int MY_HEIGHT = 163;
14      public final static int MY_WIDTH = 44;
15      //音量改变监听器
16      public interface OnVolumeChangedListener{
17          public void setYourVolume(int index);
18      }
19      //设置音量改变监听器
20      public void setOnVolumeChangeListener(OnVolumeChangedListener l){
21          mOnVolumeChangedListener = l;
```

```
22      }
23      //构造函数
24      public SoundView(Context context, AttributeSet attrs, int defStyle){
25          super(context, attrs, defStyle);
26          mContext = context;
27          //带属性、样式初始化界面
28          init();
29      }
30      //构造函数
31      public SoundView(Context context, AttributeSet attrs) {
32          super(context, attrs);
33          mContext = context;
34          //带属性初始化界面
35          init();
36      }
37      //构造函数
38      public SoundView(Context context) {
39          super(context);
40          mContext = context;
41          //不带属性、样式初始化界面
42          init();
43      }
44      //初始化
45      private void init(){
46          bm = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.sound_line);
47          bm1 = BitmapFactory.decodeResource(mContext.getResources(),
            R.drawable.sound_line1);
48          //取得图片的高宽
49          bitmapWidth = bm.getWidth();
50          bitmapHeight = bm.getHeight();
51          AudioManager am = (AudioManager)
            mContext.getSystemService(Context.AUDIO_SERVICE);
52          //设置当前音量
53          setIndex(am.getStreamVolume(AudioManager.STREAM_MUSIC));
54      }
55      //触摸事件
56      @Override
57      public boolean onTouchEvent(MotionEvent event) {
58          //根据触摸位置决定音量大小
59          int y = (int) event.getY();
60          int n = y * 15 / MY_HEIGHT;
61          setIndex(15-n);
62          return true;
63      }
64      @Override
65      protected void onDraw(Canvas canvas) {
66          int reverseIndex = 15 - index;
67          //绘制彩色音量条
68          for(int i = 0;i!=reverseIndex;++i){
69              canvas.drawBitmap(bm1, new Rect(0,0,bitmapWidth,
                bitmapHeight),
70                      new Rect(0,i*HEIGHT,bitmapWidth,i*HEIGHT+bitmap
                        Height), null);
71          }
72          //绘制灰色音量条
73          for(int i = reverseIndex;i!=15;++i){
74              canvas.drawBitmap(bm, new Rect(0,0,bitmapWidth,bi
                tmapHeight),
```

```
75                  new Rect(0,i*HEIGHT,bitmapWidth,i*HEIGHT+
                   bitmapHeight), null);
76          }
77          super.onDraw(canvas);
78      }
79      //改变音量大小
80      private void setIndex(int n){
81          if(n>15){
82              n = 15;
83          }
84          else if(n<0){
85              n = 0;
86          }
87          if(index!=n){
88              index = n;
89              if(mOnVolumeChangedListener!=null){
90                  mOnVolumeChangedListener.setYourVolume(n);
91              }
92          }
93          //重绘界面
94          invalidate();
95      }
96  }
```

（8）播放列表界面的实现代码如下所示，首先获取 VideoPlayerActivity 的 playList 作为播放列表的数据源，接着在列表中显示获取到的视频文件，并设置监听器。当单击指定元素时，将发送一个 intent 到 VideoPlayerActivity，把视频的 id 信息传递过去。

```
01  //视频选择界面
02  public class VideoChooseActivity extends Activity{
03      //视频播放列表
04      private LinkedList<MovieInfo> mLinkedList;
05      private LayoutInflater mInflater;
06      View root;
07
08      @Override
09      protected void onCreate(Bundle savedInstanceState) {
10          // TODO Auto-generated method stub
11          super.onCreate(savedInstanceState);
12          getWindow().requestFeature(Window.FEATURE_NO_TITLE);
13          setContentView(R.layout.dialog);
14          //取得视频播放列表
15          mLinkedList = VideoPlayerActivity.playList;
16          mInflater = getLayoutInflater();
17          //“返回”按钮
18          ImageButton iButton = (ImageButton) findViewById(R.id.cancel);
19          iButton.setOnClickListener(new OnClickListener(){
20              @Override
21              public void onClick(View arg0) {
22                  //关闭当前界面
23                  VideoChooseActivity.this.finish();
24              }
25          });
26          //取得播放列表界面
27          ListView myListView = (ListView) findViewById(R.id.list);
28          //设置适配器
29          myListView.setAdapter(new BaseAdapter(){
30              //取得数量
```

```
31              @Override
32              public int getCount() {
33                  // TODO Auto-generated method stub
34                  return mLinkedList.size();
35              }
36              //取得元素
37              @Override
38              public Object getItem(int arg0) {
39                  // TODO Auto-generated method stub
40                  return arg0;
41              }
42              //取得元素 id
43              @Override
44              public long getItemId(int arg0) {
45                  // TODO Auto-generated method stub
46                  return arg0;
47              }
48              //取得视图
49              @Override
50              public View getView(int arg0, View convertView, ViewGroup
                 arg2){
51                  // TODO Auto-generated method stub
52                  if(convertView==null){
53                      convertView = mInflater.inflate(R.layout.list, null);
54                  }
55                  //显示视频文件名称
56                  TextView text = (TextView) convertView.findViewById
                    (R.id.text);
57                  text.setText(mLinkedList.get(arg0).displayName);
58                  return convertView;
59              }
60          });
61          //设置按键监听器
62          myListView.setOnItemClickListener(new OnItemClickListener(){
63              //按键被按下
64              @Override
65              public void onItemClick(AdapterView<?> arg0, View arg1, int
                 arg2,
66                      long arg3) {
67                  //发送 intent 给主界面，告知要播放的视频文件信息
68                  Intent intent = new Intent();
69                  intent.putExtra("CHOOSE", arg2);
70                  VideoChooseActivity.this.setResult(Activity.RESULT_OK,
                   intent);
71                  VideoChooseActivity.this.finish();
72                  }
73          });
74      }
75  }
```

19.4　知识拓展

本章开发视频主要围绕 MediaPlayer 的使用展开，因此开发前我们要了解 MediaPlayer 相关的知识，下面我们对 MediaPlayer 的一些基本操作进行一个归纳。

（1）如何获得 MediaPlayer 实例

可以使用直接 new 的方式：MediaPlayer mp = new MediaPlayer()，也可以使用 create 的方式，如：MediaPlayer mp = MediaPlayer.create(this, R.raw.test)，这时就不用调用 setDataSource 了。

（2）如何设置要播放的文件

MediaPlayer 要播放的文件主要包括 3 个来源：

- 用户在应用中事先自带的 resource 资源，如 MediaPlayer.create(this, R.raw.test)；
- 存储在 SD 卡或其他文件，路径下的媒体文件，如 mp.setDataSource("/sdcard/test.mp3")；
- 网络上的媒体文件如 mp.setDataSource("http://www.citynorth.cn/music/confucius.mp3")。

MediaPlayer 的 setDataSource 一共有 4 个方法：

- setDataSource (String path)；
- setDataSource (FileDescriptor fd)；
- setDataSource (Context context, Uri uri)；
- setDataSource (FileDescriptor fd, long offset, long length)。

（3）对播放器的主要控制方法

Android 通过控制播放器的状态的方式来控制媒体文件的播放，其中 prepare()和 prepareAsync()提供了同步和异步两种方式设置播放器进入 prepare 状态。需要注意的是，如果 MediaPlayer 实例是由 create 方法创建的，那么第一次启动播放器不需要再调用 prepare()，因为 create()方法里已经调用过了。

start()是真正启动文件播放的方法，pause()和 stop()比较简单，起到暂停和停止播放的作用。seekTo()是定位方法，可以让播放器从指定的位置开始播放，需要注意的是该方法是异步方法，也就是说该方法返回时并不意味着定位完成，尤其是播放网络文件时，真正定位完成时会触发 OnSeekComplete.onSeekComplete()，如果需要是可以调用 setOnSeekCompleteListener(OnSeekCompleteListener)设置监听器来处理的。

release()可以释放播放器占用的资源，一旦确定不再使用播放器时应当尽早调用它释放资源。reset()可以使播放器从 Error 状态中恢复过来，重新回到 Idle 状态。

（4）设置播放器的监听器

MediaPlayer 提供了一些设置不同监听器的方法来更好地对播放器的工作状态进行监听，以及时处理各种情况，如 setOnCompletionListener(MediaPlayer.OnCompletionListener listener)、setOnErrorListener(MediaPlayer.OnErrorListener listener)等。设置播放器时需要考虑到播放器可能出现的情况，设置好监听和处理逻辑，以保持播放器的健壮性。

19.5　本章小结

本章介绍了基于系统本身播放类库的视频播放器的开发，因此读者在开发前需要对该类库的使用有一个充分的了解，包括函数如何调用以及调用的顺序。在开发过程中需要注意及时对不适用的资源进行回收，以免占用过多系统内存或者影响系统其他程序的使用。通过本章的学习，读者可以熟练掌握 mediaplayer 库的使用，了解开发播放器的流程和要点。

第 5 篇　Android 游戏开发实战案例

第 20 章　小兔跳铃铛

小兔跳铃铛是一款经典的小游戏，游戏中小兔通过踩着铃铛不断往上跳，进而不断累积积分。优美的背景音乐，可爱的小兔，简单的操作，让人在游戏中体验冬日的静谧和美丽。这款游戏大都以 Flash 的格式出现，本章将开发一个 Android 版的小兔跳铃铛。

20.1　功 能 分 析

开发游戏之前最重要的一点就是弄清楚游戏的流程，并建立游戏的逻辑。游戏的开始，首先出现的是引导界面，包括游戏介绍和操作说明，接着是游戏的声音配置，最后正式开始游戏。

如图 20.1 所示，游戏的开始小兔出现在场景的下方，背景图片是一张雪地星空图，天空中随机放着几个铃铛，游戏右下角有一个向上的按钮。由此我们应该清楚在初始化游戏界面时需要做的事情，那就是初始化前面提到过的所有界面元素。

游戏的逻辑应该是这样，当我们单击右下角的按钮时，小兔开始往上跳，通过触摸点决定小兔跳跃的方向，小兔碰到铃铛之后有一个往上的速度，同时铃铛消失。当小兔往上跳跃超过屏幕 1/2 以上时屏幕背景开始往下拖动，让人感觉到小兔在不断往上跳，同时在新出现的背景中要填充上铃铛。

游戏过程为了增加趣味性，还加入了小鸟，当小兔踩到小鸟时，当前的积分即翻倍。若小兔没能持续地踩到铃铛，则小兔会不断落下，最终到达地面，游戏结束，弹出结束画面如图 20.2 所示。



图 20.1　游戏主界面

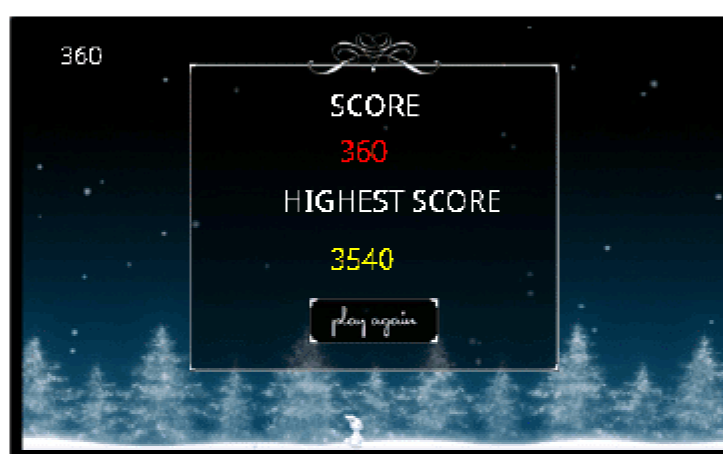


图 20.2　游戏结束画面

20.2　游戏角色设计

从上面的分析我们可以看出游戏的角色有 3 个，分别为小兔、铃铛和小鸟，下面我们

要介绍这 3 个角色类的设计。

20.2.1　小兔类

小兔作为游戏的主角，需要与其他相关的角色发生关系，如小兔与铃铛的撞击，小兔与小鸟的撞击等。此外，小兔的动作变化也比较多样，可以向左运动，可以向右运动，可以在地面，可以在空中，可以向上跳跃，可以往下掉落等等。以上这些都是我们在编写小兔类时需要考虑的问题。

如下所示，新建小兔类 Rabit.java，在类的开始声明小兔相关的成员变量，如代表小兔位置的小兔中心坐标，以及小兔的各个朝向的速度，小兔的运动状态和朝向等等。在构造函数中初始化小兔的状态为在地面，面朝右，并调用 initBitmaps 将小兔相关的图片资源载入到图片数组 bitmaps 中。

函数 isHitBell()和 isHitBird()分别用来判断小兔是否与铃铛和小鸟相撞，通过判断这两者的中心坐标的距离是否小于 20 来确定它们相撞与否。

```
001 //小兔类
002 public class Rabit {
003     //上下文
004     private Context context;
005     //用户触摸点的 x 坐标
006     private float x;
007     //用户触摸点的 y 坐标
008     private float y;
009     //小兔的中心坐标
010     private float center_x;
011     private float center_y;
012     //当前的小兔图像
013     private Bitmap currentBitmap;
014     //小兔所有图像数组
015     private Bitmap[] allBitmaps;
016     //向左的速度
017     private float speed_x_left;
018     //向右的速度
019     private float speed_x_right;
020     //向上的速度
021     private float speed_y_up;
022     //向下的速度
023     private float speed_y_down;
024     //小兔运动的目标 x 坐标
025     private float x_destination;
026     //图片的状态
027     private int pic_state;
028     //小兔的朝向
029     private int face_state;
030     //小兔在地面的状态
031     private int ground_state;
032     //小兔在空中的状态
033     private int air_state;
034     private Bitmap[] bitmaps;
035     //小兔的宽高
036     public static final float RABIT_WIDTH = Constant.RABIT_WIDTH;
```

```
037     public static final float RABIT_HEIGHT = Constant.RABIT_HEIGHT;
038
039     //rabit 图片的状态
040     public static final int RABIT_PIC_LEFT_STOP = 0;
041     public static final int RABIT_PIC_RIGHT_STOP = 1;
042     //在地面向左跳跃
043     public static final int RABIT_PIC_ON_GROUND_LEFT_JUMP0 = 2;
044     public static final int RABIT_PIC_ON_GROUND_LEFT_JUMP1 = 3;
045     //在地面向右跳跃
046     public static final int RABIT_PIC_ON_GROUND_RIGHT_JUMP0 = 4;
047     public static final int RABIT_PIC_ON_GROUND_RIGHT_JUMP1 = 5;
048     //在空中向左跳跃
049     public static final int RABIT_PIC_ON_AIR_LEFT_JUMP = 6;
050     //在空中向右跳跃
051     public static final int RABIT_PIC_ON_AIR_RIGHT_JUMP = 7;
052     //在空中面向左停止
053     public static final int RABIT_PIC_ON_AIR_LEFT_STOP = 8;
054     //在空中面向右停止
055     public static final int RABIT_PIC_ON_AIR_RIGHT_STOP = 9;
056     //在空中向左下角运动
057     public static final int RABIT_PIC_ON_AIR_LEFT_DOWN = 10;
058     //在空中向右下角运动
059     public static final int RABIT_PIC_ON_AIR_RIGHT_DOWN = 11;
060     //rabit 朝左朝右状态
061     public static final int RABIT_FACE_LEFT = 1;
062     public static final int RABIT_FACE_RIGHT = 2;
063     //rabit 在地面状态
064     public static final int RABIT_NOT_ON_GROUND = 0;
065     public static final int RABIT_LEFT_STOP = 1;
066     public static final int RABIT_RIGHT_STOP = 2;
067     public static final int RABIT_LEFT_MOVE1_ON_GROUND = 3;
068     public static final int RABIT_LEFT_MOVE2_ON_GROUND = 4;
069     public static final int RABIT_RIGHT_MOVE1_ON_GROUND = 5;
070     public static final int RABIT_RIGHT_MOVE2_ON_GROUND = 6;
071     //rabit 在空中的状态
072     public static final int RABIT_ON_AIR_UP0 = 0;
073     public static final int RABIT_ON_AIR_UP1 = 1;
074     public static final int RABIT_ON_AIR_UP2 = 2;
075     public static final int RABIT_ON_AIR_UP3 = 3;
076     public static final int RABIT_ON_AIR_UP4 = 4;
077     public static final int RABIT_ON_AIR_UP5 = 5;
078     public static final int RABIT_ON_AIR_STOP = 6;
079     public static final int RABIT_ON_AIR_DOWN = 7;
080
081     //rabit 每次跳跃刷新的高度是不同的
082     public static final float RABIT_UP_DESTIATON0 = 30;
083     public static final float RABIT_UP_DESTIATON1 = 20;
084     public static final float RABIT_UP_DESTIATON2 = 10;
085     //rabit 每次下降刷新的距离是不同的
086     public static final float RABIT_DOWN_DESTIATON0 = 10;
087     public static final float RABIT_DOWN_DESTIATON1 = 20;
088     public static final float RABIT_DWON_DESTIATON2 = 30;
089     public static final float SPEED_X = 10;
090     public static final float SPEED_Y = 15;
091     public static final float SPEED_X_ON_AIR = 15;
092     private Paint paint;
093
094     public Rabit(Context context){
095        this.context = context;
```

```
096         //初始化图片资源
097         initBitmaps();
098         //设置小兔的初始位置
099         this.setX(Constant.RABIT_INIT_X);
100         this.setY(Constant.RABIT_INIT_Y);
101         x_destination = x;
102         //初始化速度
103         speed x left = 0;
104         speed x right = 0;
105         speed y down = 15;
106         speed y up = 15;
107         paint = new Paint();
108         //初始化小兔的状态
109         ground_state = RABIT_RIGHT_STOP;
110         air_state = RABIT_ON_AIR_UP0;
111         face_state = RABIT_FACE_RIGHT;
112         this.pic_state = RABIT_PIC_RIGHT_STOP;
113
114     }
115     //初始化函数
116     public void init(){
117         //初始化小兔的坐标和运动状态
118         this.setY(Constant.RABIT INIT Y);
119         ground state = RABIT RIGHT STOP;
120         air state = RABIT ON AIR UP0;
121         face state = RABIT FACE RIGHT;
122         this.pic state = RABIT PIC RIGHT STOP;
123     }
124     //初始化图片资源
125     private void initBitmaps(){
126         bitmaps = new Bitmap[12];
127         bitmaps[Rabit.RABIT_PIC_LEFT_STOP] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.rabit left stop);
128         bitmaps[Rabit.RABIT PIC RIGHT STOP] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.rabit right stop);
129         bitmaps[RABIT PIC ON GROUND LEFT JUMP0] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.rabit on ground left jump0);
130         bitmaps[RABIT PIC ON GROUND LEFT JUMP1] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.rabit on ground left jump1);
131         bitmaps[RABIT PIC ON GROUND RIGHT JUMP0] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.rabit on ground right jump0);
132         bitmaps[RABIT PIC ON GROUND RIGHT JUMP1] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.rabit on ground right jump1);
133         bitmaps[RABIT_PIC_ON_AIR_LEFT_JUMP] =
            bitmaps[RABIT_PIC_ON_GROUND_LEFT_JUMP0];
134         bitmaps[RABIT_PIC_ON_AIR_RIGHT_JUMP] =
            bitmaps[RABIT_PIC_ON_GROUND_RIGHT_JUMP0];
135         bitmaps[RABIT_PIC_ON_AIR_LEFT_DOWN] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.rabit_on_air_left_down);
136         bitmaps[RABIT_PIC_ON_AIR_RIGHT_DOWN] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.rabit on air right down);;
137         bitmaps[RABIT_PIC_ON_AIR_LEFT_STOP] =
```

```
        BitmapFactory.decodeResource(context.getResources(),
        R.drawable.rabit on air left stop);;
138     bitmaps[RABIT PIC ON AIR RIGHT STOP] =
        BitmapFactory.decodeResource(context.getResources(),
        R.drawable.rabit on air right stop);;
139   }
140   //绘制图片
141   public void onDraw(Canvas canvas){
142       canvas.drawBitmap(bitmaps[pic_state], x, y, paint);
143   }
144   //判断是否撞到铃铛
145   public boolean isHitBell(Bell bell){
146       if(bell.isExplode()) return false;
147       //获得小兔的中心坐标
148       float x1 = center_x;
149       float y1 = center_y;
150       //获得铃铛的中心坐标
151       float x2 = bell.getCenter x();
152       float y2 = bell.getCenter y();
153       //两个中心坐标的距离小于 20 则表明撞到
154       return Math.pow(x1-x2, 2)+Math.pow(y1-y2, 2) < 400;
155   }
156   //判断是否撞到铃铛
157   public boolean isHitBird(Bird bird){
158       //获得小兔的中心坐标
159       float x1 = center x;
160       float y1 = center_y;
161       //获得小鸟的中心坐标
162       float x2 = bird.getCenter x();
163       float y2 = bird.getCenter y();
164       //两个中心坐标的距离小于 20 则表明撞到
165       return Math.pow(x1-x2, 2)+Math.pow(y1-y2, 2) < 400;
166   }
```

如下所示，是小兔类中用来对成员变量进行读取和设置的函数，包括前面提到的所有的成员变量，通过这些函数我们可以获取小兔所有相关的状态信息。

```
001 //判断小兔是否在地面
002 public boolean isOnGround(){
003     return !(ground_state==RABIT_NOT_ON_GROUND);
004 }
005 //判断小兔是否在空中
006 public boolean isOnAir(){
007     return !isOnGround();
008 }
009 //判断小兔是否停在地面
010 public boolean isOnGroundStop(){
011     return ground_state==RABIT_LEFT_STOP||ground_state==
        RABIT RIGHT STOP;
012 }
013 //判断小兔是否面朝左
014 public boolean isFaceLeft(){
015     return face state==RABIT FACE LEFT;
016 }
017 //判断小兔是否面朝右
018 public boolean isFaceRight(){
019     return !isFaceLeft();
020 }
```

```
021 //取得上下文
022 public Context getContext() {
023     return context;
024 }
025 //设置上下文
026 public void setContext(Context context) {
027     this.context = context;
028 }
029 //取得用户触摸点的 X 坐标
030 public float getX() {
031     return x;
032 }
033 //设置用户触摸点的 X 坐标
034 public void setX(float x) {
035     this.x = x;
036     this.center x = x + RABIT WIDTH/2;
037 }
038 //取得用户触摸点的 Y 坐标
039 public float getY() {
040     return y;
041 }
042 //设置用户触摸点的 Y 坐标
043 public void setY(float y) {
044     this.y = y;
045     this.center y = y + RABIT HEIGHT/2;
046 }
047 //取得当前小兔的图片
048 public Bitmap getCurrentBitmap() {
049     return currentBitmap;
050 }
051 //设置当前小兔的图片
052 public void setCurrentBitmap(Bitmap currentBitmap) {
053     this.currentBitmap = currentBitmap;
054 }
055 //取得图片组
056 public Bitmap[] getAllBitmaps() {
057     return allBitmaps;
058 }
059 //设置图片组
060 public void setAllBitmaps(Bitmap[] allBitmaps) {
061     this.allBitmaps = allBitmaps;
062 }
063 //取得向左的速度
064 public float getSpeed_x_left() {
065     return speed_x_left;
066 }
067 //设置向左的速度
068 public void setSpeed x left(float speedXLeft) {
069     speed x left = speedXLeft;
070 }
071 //取得向右的速度
072 public float getSpeed_x_right() {
073     return speed_x_right;
074 }
075 //设置向右的速度
076 public void setSpeed x right(float speedXRight) {
077     speed x right = speedXRight;
078 }
079 //取得向上的速度
```

```
080 public float getSpeed y up() {
081     return speed y up;
082 }
083 //设置向上的速度
084 public void setSpeed_y_up(float speedYUp) {
085     speed_y_up = speedYUp;
086 }
087 //取得向下的速度
088 public float getSpeed y down() {
089     return speed y down;
090 }
091 //设置向下的速度
092 public void setSpeed_y_down(float speedYDown) {
093     speed_y_down = speedYDown;
094 }
095 //取得图片集
096 public Bitmap[] getBitmaps() {
097     return bitmaps;
098 }
099 //设置图片集
100 public void setBitmaps(Bitmap[] bitmaps) {
101     this.bitmaps = bitmaps;
102 }
103 //取得画笔
104 public Paint getPaint() {
105     return paint;
106 }
107 //设置画笔
108 public void setPaint(Paint paint) {
109     this.paint = paint;
110 }
111 //取得 x 轴终点位置
112 public float getX destination() {
113     return x destination;
114 }
115 //设置 x 轴终点位置
116 public void setX_destination(float xDestination) {
117     x_destination = xDestination;
118 }
119 //取得朝向
120 public int getFace state() {
121     return face state;
122 }
123 //设置朝向
124 public void setFace_state(int faceState) {
125     face_state = faceState;
126 }
127 //取得地面状态
128 public int getGround state() {
129     return ground state;
130 }
131 //设置地面状态
132 public void setGround_state(int groundState) {
133     ground_state = groundState;
134 }
135 //取得空中状态
136 public int getAir state() {
137     return air state;
138 }
```

```
139 //设置空中状态
140 public void setAir state(int airState) {
141     air state = airState;
142 }
143 //取得图片状态
144 public int getPic_state() {
145     return pic state;
146 }
147 //设置图片状态
148 public void setPic state(int picState) {
149     pic state = picState;
150 }
151 //取得小兔的中心 X 坐标
152 public float getCenter x() {
153     return center x;
154 }
155 //设置小兔的中心 X 坐标
156 public void setCenter x(float centerX) {
157     center_x = centerX;
158     this.x = centerX - RABIT_WIDTH/2;
159 }
160 //取得小兔的中心 Y 坐标
161 public float getCenter y() {
162     return center y;
163 }
164 //设置小兔的中心 Y 坐标
165 public void setCenter_y(float centerY) {
166     center_y = centerY;
167     y = centerY - RABIT_HEIGHT/2;
168 }
```

20.2.2　铃铛类

新建铃铛类 Bell.java，代码如下所示。和小兔一样，铃铛首先需要有表示其位置的 x、y 坐标变量（如代码 04、05 行所示）。铃铛的状态大体上分为两种：正常和爆炸，而爆炸又分为 3 个阶段，代码 21～24 行定义了铃铛的所有状态。

在构造函数中通过调用函数 initBitmaps()对图片资源进行初始化，若构造函数中传入了铃铛的坐标，则对坐标也进行初始化，接着初始化画笔和铃铛的默认状态。

```
01  //铃铛类
02  public class Bell {
03      //铃铛的中心坐标
04      private float center_x;
05      private float center_y;
06      //铃铛图片组
07      private Bitmap[] bitmaps;
08      //上下文
09      private Context context;
10      //铃铛的状态
11      private int state;
12      //画笔
13      private Paint paint;
14      //铃铛通常情况下的宽高
15      public static final float BELL OK WIDTH = Constant.BELL OK WIDTH;
16      public static final float BELL_OK_HEIGHT= Constant.BELL_OK_HEIGHT;
```

```
17      //铃铛爆炸情况下的宽高
18      public static final float BELL_EXPLODE_WIDTH =
        Constant.BELL_EXPLODE_WIDTH;
19      public static final float BELL_EXPLODE_HEIGHT =
        Constant.BELL_EXPLODE_HEIGHT;
20      //铃铛的状态
21      public static final int BELL_OK = 0;
22      public static final int BELL_EXPLODE0 = 1;
23      public static final int BELL_EXPLODE1 = 2;
24      public static final int BELL_EXPLODE2 = 3;
25
26      //构造函数，带铃铛坐标
27      public Bell(Context context, float center_x, float center_y) {
28          //初始化中心坐标
29          this.center_x = center_x;
30          this.center_y = center_y;
31          //初始化上下文
32          this.context = context;
33          //初始化铃铛图片资源
34          initBitmaps();
35          //初始化铃铛状态
36          state = BELL_OK;
37          paint = new Paint();
38      }
39      //构造函数，不带铃铛坐标
40      public Bell(Context context){
41          this.context = context;
42          initBitmaps();
43          state = BELL_OK;
44          paint = new Paint();
45      }
46      //初始化铃铛图片组
47      private void initBitmaps() {
48          bitmaps = new Bitmap[4];
49          //正常图片
50          bitmaps[0] = BitmapFactory.decodeResource(context.
            getResources(),
51                  R.drawable.bell_ok);
52          //爆炸图片
53          bitmaps[1] = BitmapFactory.decodeResource(context.
            getResources(),
54                  R.drawable.bell_explode0);
55          bitmaps[2] = BitmapFactory.decodeResource(context.
            getResources(),
56                  R.drawable.bell_explode1);
57          bitmaps[3] = BitmapFactory.decodeResource(context.
            getResources(),
58                  R.drawable.bell_explode2);
59      }
60      //绘制铃铛
61      public void onDraw(Canvas canvas) {
62          switch(state){
63          //铃铛完好时
64          case BELL_OK:
65                  canvas.drawBitmap(bitmaps[BELL_OK],
                    center_x-BELL_OK_WIDTH/2, center_y-BELL_OK_HEIGHT/2,
                    paint);
66              break;
67              //铃铛爆炸第一阶段
```

```
68          case BELL EXPLODE0:
69                  canvas.drawBitmap(bitmaps[BELL EXPLODE0],
                    center x-BELL EXPLODE WIDTH/2,
                    center y-BELL EXPLODE HEIGHT/2, paint);
70              break;
71              //铃铛爆炸第二阶段
72          case BELL_EXPLODE1:
73                  canvas.drawBitmap(bitmaps[BELL_EXPLODE1],
                    center_x-BELL_EXPLODE_WIDTH/2,
                    center_y-BELL_EXPLODE_HEIGHT/2, paint);
74              break;
75              //铃铛爆炸第三阶段
76          case BELL EXPLODE2:
77                  canvas.drawBitmap(bitmaps[BELL EXPLODE2],
                    center x-BELL EXPLODE WIDTH/2,
                    center y-BELL EXPLODE HEIGHT/2, paint);
78              break;
79              //默认情况下铃铛完好
80          default:
81                  canvas.drawBitmap(bitmaps[BELL OK],
                    center_x-BELL_OK_WIDTH/2, center_y-BELL_OK_HEIGHT/2,
                    paint);
82              break;
83          }
84      }
```

如下所示为铃铛所有相关属性的设置和获取函数，包括铃铛是否爆炸，铃铛的中心坐标和铃铛的状态等。

```
01  //铃铛是否 OK
02  public boolean isOK(){
03      return state==BELL_OK;
04  }
05  //铃铛是否爆炸
06  public boolean isExplode(){
07      return !isOK();
08  }
09  //取得铃铛的中心 X 坐标
10  public float getCenter x() {
11      return center_x;
12  }
13  //设置铃铛的中心 X 坐标
14  public void setCenter x(float centerX) {
15      center x = centerX;
16  }
17  //取得铃铛的中心 Y 坐标
18  public float getCenter y() {
19      return center_y;
20  }
21  //设置铃铛的中心 Y 坐标
22  public void setCenter y(float centerY) {
23      center y = centerY;
24  }
25  //取得图片组
26  public Bitmap[] getBitmaps() {
27      return bitmaps;
28  }
29  //设置图片组
30  public void setBitmaps(Bitmap[] bitmaps) {
```

```
31      this.bitmaps = bitmaps;
32  }
33  //取得上下文
34  public Context getContext() {
35      return context;
36  }
37  //设置上下文
38  public void setContext(Context context) {
39      this.context = context;
40  }
41  //取得图片状态
42  public int getState() {
43      return state;
44  }
45  //设置图片状态
46  public void setState(int state) {
47      this.state = state;
48  }
49  //取得画笔
50  public Paint getPaint() {
51      return paint;
52  }
53  //设置画笔
54  public void setPaint(Paint paint) {
55      this.paint = paint;
56  }
```

20.2.3　小鸟类

新建小鸟类 Bird.java 如下所示，新建小鸟的位置坐标变量 center_x 和 center_y、小鸟是否在屏幕中的标志位 on_screen，以及其他的小鸟速度变量、状态变量等等。接着在小鸟的构造函数中初始化小鸟的图片资源，并初始化小鸟的状态为向左飞行。

```
01  //小鸟类
02  public class Bird {
03      //上下文
04      private Context context;
05      //小鸟的速度
06      private float speed = Constant.BIRD_SPEED;
07      //小鸟被撞击之后的速度
08      private float speed_power = Constant.BIRD_SPEED_POWER;
09      //小鸟的中心坐标
10      private float center_x;
11      private float center_y;
12      //小鸟是否出现在屏幕
13      private boolean on_screen = false;
14      //小鸟的状态
15      private int state;
16      //小鸟图片组
17      private Bitmap[] bitmaps  = new Bitmap[8];
18      //画笔
19      private Paint paint = new Paint();
20      //小鸟的高宽
21      public static final float BIRD_WIDTH = Constant.BIRD_WIDTH;
22      public static final float BIRD_HIGHT = Constant.BIRD_HIGHT;
```

```
23      //小鸟的速度
24      public static final float BIRD_SPEED = Constant.BIRD_SPEED;
25      public static final float BIRD_SPEED_POWER =
        Constant.BIRD_SPEED_POWER;
26      //小鸟的飞行状态，向左飞行
27      public static final int BIRD_LEFT_FLY0 = 0;
28      public static final int BIRD_LEFT_FLY1 = 1;
29      public static final int BIRD_LEFT_FLY2 = 2;
30      //向右飞行
31      public static final int BIRD_RIGHT_FLY0 = 3;
32      public static final int BIRD_RIGHT_FLY1 = 4;
33      public static final int BIRD_RIGHT_FLY2 = 5;
34      //向左加速飞行
35      public static final int BIRD_LEFT_FLY_POWER = 6;
36      //向右加速飞行
37      public static final int BIRD_RIGHT_FLY_POWER = 7;
38      //构造函数
39      public Bird(Context context){
40          this.context = context;
41          //初始化小鸟的状态
42          state = BIRD_LEFT_FLY0;
43          //初始化图片组
44          init_bitmaps();
45      }
46      //初始化图片组
47      public void init_bitmaps(){
48          //向左飞行
49          bitmaps[0] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.bird_left_fly0);
50          bitmaps[1] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.bird_left_fly1);
51          bitmaps[2] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.bird_left_fly2);
52          //向右飞行
53          bitmaps[3] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.bird_right_fly0);
54          bitmaps[4] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.bird_right_fly1);
55          bitmaps[5] =
            BitmapFactory.decodeResource(context.getResources(),
            R.drawable.bird_right_fly2);
56          //向左加速飞行
57          bitmaps[6] = bitmaps[1];
58          //向右加速飞行
59          bitmaps[7] = bitmaps[4];
60      }
61      //绘制小鸟
62      public void onDraw(Canvas canvas){
63          canvas.drawBitmap(bitmaps[state], center_x-BIRD_WIDTH/2,
            center_y-BIRD_HIGHT/2, paint);
64      }
```

如下所示，函数 isHited()和 isFaceLeft()分别用来判断小鸟是否被撞击和小鸟是否朝左

飞行，其他函数则分别用来获取和设置小鸟的各种属性，包括小鸟的中心坐标、小鸟的速度、小鸟的状态、小鸟是否出现在屏幕等。

```
01  //是否被碰撞到
02  public boolean isHited(){
03      return (state == BIRD LEFT FLY POWER || state == BIRD RIGHT
        FLY POWER);
04  }
05  //是否面朝左
06  public boolean isFaceLeft(){
07      return (state == BIRD_LEFT_FLY0 || state == BIRD_LEFT_FLY1 || state
        == BIRD_LEFT_FLY2 || state == BIRD_LEFT_FLY_POWER);
08  }
09  //是否面朝右
10  public boolean isFaceRight(){
11      return !isFaceLeft();
12  }
13  //取得速度
14  public float getSpeed() {
15      return speed;
16  }
17  //设置速度
18  public void setSpeed(int speed) {
19      this.speed = speed;
20  }
21  //取得被撞击后的速度
22  public float getSpeed_power() {
23      return speed_power;
24  }
25  //获得被撞击后的速度
26  public void setSpeed power(int speedPower) {
27      speed power = speedPower;
28  }
29  //取得中心 x 坐标
30  public float getCenter_x() {
31      return center_x;
32  }
33  //设置中心 x 坐标
34  public void setCenter x(float centerX) {
35      center x = centerX;
36  }
37  //取得中心 y 坐标
38  public float getCenter_y() {
39      return center_y;
40  }
41  //设置中心 y 坐标
42  public void setCenter y(float centerY) {
43      center y = centerY;
44  }
45  //判断是否在屏幕
46  public boolean isOn_screen() {
47      return on_screen;
48  }
49  //设置是否在屏幕
50  public void setOn screen(boolean onScreen) {
51      on screen = onScreen;
52  }
53  //取得状态
```

```
54  public int getState() {
55      return state;
56  }
57  //设置状态
58  public void setState(int state) {
59      this.state = state;
60  }
```

20.3　游戏背景设计

游戏中除了各种角色外，还有很重要的一项，那就是游戏的背景。游戏的背景主要包括游戏过程中的背景图片和背景音乐。

20.3.1　背景音乐

一个好玩的游戏绝对少不了背景音乐，因此接下来我们要为游戏添加背景音乐，代码如下所示，新建 AudioProvider.java。游戏中的音乐主要分为两种，第一种是游戏的背景音乐，采用 MediaPlayer 循环播放；另外一种是游戏过程中触发的角色音乐，有铃铛碰撞的音乐和鸟叫的音乐。由于后面一种音乐比较短促，因此采用音乐池 SoundPool 进行播放，比较节约资源。

```
01  package com.supermario.rabit;                //声明包语句
02~05 行为引入相关类，这里不再列举，请阅读光盘内容
//……
06  //背景音乐提供类
07  public class AudioProvider {
08      //媒体播放器，用于播放背景音乐
09      private MediaPlayer media_player;
10      //音乐池
11      private SoundPool soundPool;
12      //撞击铃铛声音
13      private int soundID_ding;
14      //鸟叫声
15      private int soundID_twitter;
16      //音量
17      private float volume;
18      //构造函数
19      public AudioProvider(Context context) {
20          //加载背景音乐
21          media_player = MediaPlayer.create(context, R.raw.bg);
22          media_player.setLooping(true);
23          //使用音乐池播放短小的音乐
24          soundPool = new SoundPool(2, AudioManager.STREAM_MUSIC, 0);
25          soundID_ding = soundPool.load(context, R.raw.ding,1 );
26          soundID_twitter = soundPool.load(context, R.raw.twitter, 1);
27          //取得当前音量大小
28          AudioManager mgr = (AudioManager) context.getSystemService(
29                  Context.AUDIO_SERVICE);
30          //当前音量
31          float streamVolumeCurrent = mgr
32                  .getStreamVolume(AudioManager.STREAM_MUSIC);
```

```
33          //最大音量
34          float streamVolumeMax = mgr
35                  .getStreamMaxVolume(AudioManager.STREAM_MUSIC);
36          volume = streamVolumeCurrent / streamVolumeMax;
37      }
38      //播放铃声
39      public void play_bell_ding(){
40          soundPool.play(soundID_ding, volume, volume, 1, 0, 1);
41      }
42      //播放鸟叫
43      public void play_twitter(){
44          soundPool.play(soundID_twitter, volume, volume, 1, 0, 1);
45      }
46      //播放背景音乐
47      public void play_bg(){
48          media_player.start();
49      }
50      //释放媒体资源
51      public void release(){
52          if(media_player.isPlaying())
53              media_player.stop();
54          media_player.release();
55          soundPool.release();
56      }
57  }
```

20.3.2　背景图片

背景图片控制类可以说是本程序控制的最核心也是最难的部分，因为涉及到图片显示位置的算法。有几个值决定着这个算法，一个是屏幕的分辨率，本程序使用的屏幕分辨率是 400×240，一个是背景图片的分辨率，本程序使用的背景图片分辨率是 400×720。同时为了制造出小兔在地上和在空中上升的感觉，要根据小兔的高度缓慢拖动背景图片。因为背景图片的高度是固定的，而小兔的高度可能超过背景图片的高度，因此超过的部分需要利用算法进行拼凑和转换。

如下所示，代码 25～27 行利用算法将小兔的运动位置分成 3 种情况，分别显示相应的背景图片。函数 drag_up()和 drap_down()分别用来设置小兔当前的高度，进而控制背景图片的显示。函数 isLowest()用来判断小兔是否已经掉落到最底部。

```
01  package com.supermario.rabit;                //声明包语句
02~06 行为引入相关类，这里不再列举，请阅读光盘内容
//……
07  //背景图片设置类
08  public class Backgroud {
09      private Context context;
10      //背景图片
11      private Bitmap bg;
12      //画笔
13      private Paint paint;
14      private int my_y;
15      //构造函数
16      public Backgroud(Context context){
17          this.context = context;
```

```
18          paint = new Paint();
19          //初始化图片资源
20          bg = BitmapFactory.decodeResource(context.getResources(),
            .drawable.bg);
21          //小兔的高度
22          my_y  = 0;
23      }
24      //绘制背景
25      public void onDraw(Canvas canvas){
26          if(my_y >= 0 && my_y <= 480){
27              //图片的高度为 720，界面高度为 240，
28              //这样可以让图片底部刚好与界面底部平齐
29              int screen_y = my_y - 480;
30              canvas.drawBitmap(bg, 0, screen_y, paint);
31          }else{
32              //当高度超过 480
33              int y_up = my_y % 480;
34              int y_down = (my_y-240) % 480;
35              if(y_up>=y_down){
36                  int screen_y = y_up - 480;
37                  //绘制一张背景图
38                  canvas.drawBitmap(bg, 0, screen_y, paint);
39              }else{
40                  //绘制两张背景图进行重叠
41                  int screen_y1 = y_up - 480;
42                  int screent_y2 = y_up;
43                  canvas.drawBitmap(bg, 0, screen_y1, paint);
44                  canvas.drawBitmap(bg, 0, screent_y2, paint);
45              }
46          }
47      }
48      //初始化小兔高度
49      public void init(){
50          this.my_y = 0;
51      }
52      //背景往下拉
53      public void drag_down(int px){
54          this.my_y += px;
55      }
56      //背景往上拉
57      public void drag_up(int px){
58          //为了使下降的落地的时间不至于过长，剪掉了兔子上升的背景距离
59          if(my_y > 960){
60              my_y = (my_y - 480) % 480 + 480;
61          }
62          this.my_y -= px;
63      }
64      //判断小兔是否掉落到最低点
65      public boolean isLowest(){
66          return (my_y >= -5 && my_y <= 10);
67      }
68      //取得上下文
69      public Context getContext() {
70          return context;
71      }
72      //设置上下文
73      public void setContext(Context context) {
74          this.context = context;
75      }
```

```
76      //取得画笔
77      public Paint getPaint() {
78          return paint;
79      }
80      //设置画笔
81      public void setPaint(Paint paint) {
82          this.paint = paint;
83      }
84  }
```

20.4　游戏辅助界面

游戏进行过程中除了游戏互动界面外还有一些辅助界面，比如一开始的游戏说明界面，游戏设置界面和游戏结束界面。这些辅助界面与游戏息息相关，通过这些界面用户可以更快地熟悉游戏，并根据自己的喜好定制游戏过程。

20.4.1　开场画面

如图 20.3 所示为游戏开场界面，实现代码也很简单，如下所示，在构造函数中获得图片资源并在 onDraw()函数中将图片绘制出来。



图 20.3　游戏开场界面

```
01  package com.supermario.rabit;                        //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容
//……
08  //游戏开场画面
09  public class IntroduceView extends View{
10      private Bitmap bg;
11      private Paint paint;
12      //新建画笔，获得图片
13      public IntroduceView(Context context) {
14          super(context);
15          paint = new Paint();
16          bg=BitmapFactory.decodeResource(this.getResources(),
            R.drawable.introduce);
```

```
17      }
18      //将图片绘制到画布上
19      protected void onDraw(Canvas canvas) {
20          super.onDraw(canvas);
21          canvas.drawBitmap(bg, 0, 0, paint);
22      }
23  }
24
```

20.4.2　帮助界面

如图 20.4 所示为游戏帮助界面，代码如下所示，也是在构造函数中初始化图片资源然后显示到界面中。

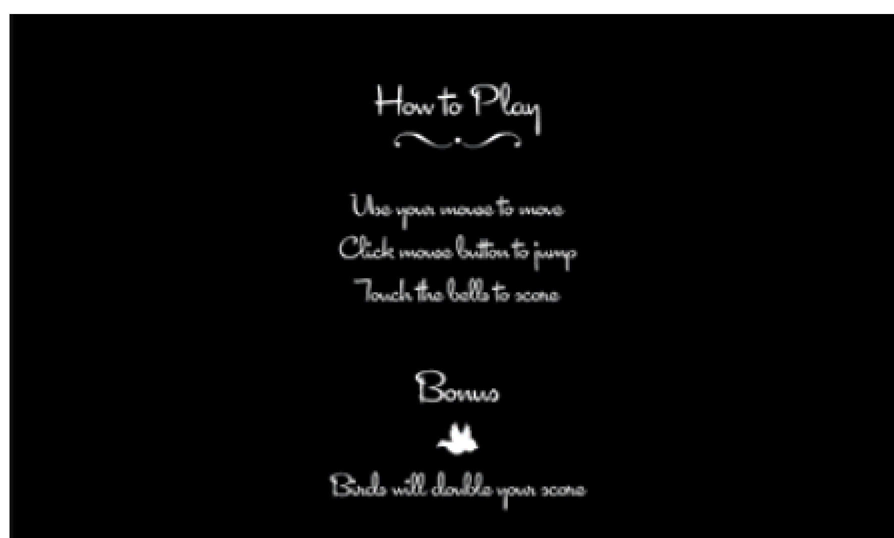


图 20.4　游戏帮助画面

```
01  package com.supermario.rabit;                //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容
//……
08  //游戏帮助界面
09  public class HelpView extends View{
10      private Bitmap bg;
11      private Paint paint;
12      //初始化画笔，获得图片资源
13      public HelpView(Context context) {
14          super(context);
15          // TODO Auto-generated constructor stub
16          paint = new Paint();
17          bg = BitmapFactory.decodeResource(getResources(),
            R.drawable.help);
18      }
19      //在画布上绘制图片
20      protected void onDraw(Canvas canvas){
21          canvas.drawBitmap(bg, 0, 0, paint);
22      }
23  }
```

20.4.3　声音设置界面

如图 20.5 所示为声音设置界面，该界面有两个按钮，可以控制游戏中是否开启音乐。实现代码如下所示，在该界面的核心函数 onDraw()中根据变量 click 和 audio_on 对字体颜色进行控制。当单击了字体所在位置的区域时，将对应位置的字体颜色变为红色，表示选中。



图 20.5　声音设置界面

```
01  package com.supermario.rabit;                        //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容
//……
08  //声音设置界面
09  public class AudioView extends View {
10      //画笔
11      Paint paint;
12      //默认声音关闭
13      boolean audio_on = false;
14      //默认未单击
15      boolean click = false;
16      public AudioView(Context context) {
17          super(context);
18          paint = new Paint();
19      }
20      //绘制图片
21      public void onDraw(Canvas canvas){
22          super.onDraw(canvas);
23          //将画布背景设为黑色
24          canvas.drawColor(Color.BLACK);
25          paint.setTextSize(20);
26          //字体颜色为白色
27          paint.setColor(Color.WHITE);
28          //绘制文字
29          canvas.drawText("是否打开音效?", 125, 85, paint);
30          if(click){
31              if(audio_on){
32                  //若打开声音则将"是"设为红色
33                  paint.setColor(Color.RED);
34                  canvas.drawText("是", 33, 197, paint);
35                  paint.setColor(Color.WHITE);
36                  canvas.drawText("否", 336, 197, paint);
37              }else{
38                  //若关闭声音则将"否"设为红色
39                  paint.setColor(Color.WHITE);
40                  canvas.drawText("是", 33, 197, paint);
41                  paint.setColor(Color.RED);
42                  canvas.drawText("否", 336, 197, paint);
43              }
44          }else {
45                  //未单击则将两者都设为白色
46                  paint.setColor(Color.WHITE);
47                  canvas.drawText("是", 33, 197, paint);
48                  canvas.drawText("否", 336, 197, paint);
```

```
49              }
50          }
51          //返回声音是否打开
52          public boolean isAudio_on() {
53              return audio_on;
54          }
55          //设置声音是否打开
56          public void setAudio_on(boolean audioOn) {
57              audio_on = audioOn;
58          }
59          //返回是否单击到
60          public boolean isClick() {
61              return click;
62          }
63          //设置是否单击到
64          public void setClick(boolean click) {
65              this.click = click;
66          }
67      }
```

20.4.4　结束界面

如图 20.6 所示，是游戏结束后显示的界面，通过画笔和画布在屏幕中间绘制一个显示分数的界面。代码 28～61 行设置了各个部分文字的字体大小和颜色，并根据是否单击了 play again 按钮，在该按钮位置显示不同的图片，以标识是否单击了该按钮。



图 20.6　结束界面

```
01  package com.supermario.rabit;                              //声明包语句
02~06 为引入相关类，这里不再列举，请阅读光盘内容
//……
07  //游戏结束界面
08  public class Conclusion {
09      //是否按下按钮
10      private boolean pressed = false;
11      private GameSurfaceView context;
12      //背景图片
13      private Bitmap bg;
14      //答复按钮
15      private Bitmap replay_button;
16      private Paint paint = new Paint();
17
18      public Conclusion(GameSurfaceView context) {
19          this.context = context;
```

```
20          //获得图片资源
21          this.bg = BitmapFactory.decodeResource(context.getContext()
22                  .getResources(), R.drawable.conclusion);
23          //获得按钮图片
24          replay_button = BitmapFactory.decodeResource
            (context.getContext()
25                  .getResources(), R.drawable.replay_button);
26      }
27      //绘制图片
28      public void onDraw(Canvas canvas) {
29          if (context.getHighest score() > context.getScore()) {
30              //绘制背景图片
31              canvas.drawBitmap(bg, 95, 0, paint);
32              //设置字体颜色为白色
33              paint.setColor(Color.WHITE);
34              paint.setTextSize(17);
35              canvas.drawText("SCORE", 175, 51, paint);
36              //分数以红色字体标出
37              paint.setColor(Color.RED);
38              canvas.drawText("" + context.getScore(), 180, 77, paint);
39              paint.setColor(Color.WHITE);
40              //最高分
41              canvas.drawText("HIGHEST SCORE", 148, 105, paint);
42              //黄色字体标出最高分
43              paint.setColor(Color.YELLOW);
44              canvas.drawText(""+context.getHighest score(), 175, 138,
                paint);
45
46              if (pressed) {
47                  canvas.drawBitmap(replay button, 164, 156, paint);
48              }
49          } else {
50              canvas.drawBitmap(bg, 95, 0, paint);
51              paint.setColor(Color.RED);
52              paint.setTextSize(20);
53              //使用红色字体显示
54              canvas.drawText("CONGRATULATIONS!", 100, 70, paint);
55              paint.setColor(Color.WHITE);
56              paint.setTextSize(17);
57              //破纪录时用白色字体显示最高分
58              canvas.drawText("New Hightest Score:", 125, 115, paint);
59              canvas.drawText(""+context.getScore(), 180, 147, paint);
60          }
61      }
62      //初始化单击标志位
63      public void init() {
64          pressed = false;
65      }
66      //获得是否已单击
67      public boolean isPressed() {
68          return pressed;
69      }
70      //设置是否已单击
71      public void setPressed(boolean pressed) {
```

```
72          this.pressed = pressed;
73      }
74  }
```

20.5　游戏过程

（1）游戏一开始将启动 RabitActivity，实现代码如下所示，在 onCreate()函数中执行 initWindow()函数初始化界面，并将当前界面设置为开场界面。在 onTouchEvent()函数中根据当前界面执行不同的操作。若当前在开场界面，则单击界面将跳转到帮助界面，同样在帮助界面将跳转到声音设置界面。在声音设置界面根据单击的位置判断用户是单击了“是”还是“否”，进而设置声音开关，并进入 GameActivity。

```
001 package com.supermario.rabit;                //声明包语句
002~011 行为引入相关类，这里不再列举，请阅读光盘内容
//……
012 public class RabitActivity extends Activity {
013     //游戏开场画面
014     private IntroduceView introduceView;
015     //游戏帮助界面
016     private HelpView helpView;
017     //音效打开关闭设置界面
018     private AudioView audioView;
019     private View currentView;
020     //默认关闭音效
021     private boolean audio on = false;
022     //构造函数
023     public void onCreate(Bundle savedInstanceState) {
024         super.onCreate(savedInstanceState);
025         //开场界面
026         introduceView = new IntroduceView(this);
027         //帮助界面
028         helpView = new HelpView(this);
029         //声音设置界面
030         audioView = new AudioView(this);
031         //初始化界面
032         initWindow();
033         //设置当前显示界面为开场界面
034         setContentView(introduceView);
035         //当前界面为开场界面
036         this.currentView = introduceView;
037     }
038     @Override
039     protected void onRestart() {
040         // TODO Auto-generated method stub
041         super.onRestart();
042         this.finish();
043     }
044     //初始化界面
045     private void initWindow() {
046         //设置全屏
047         requestWindowFeature(Window.FEATURE_NO_TITLE);
048         this.getWindow().setFlags(WindowManager.LayoutParams.
            FLAG_FULLSCREE,
049                 WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

```
050     }
051     //进入游戏界面
052     public void toGameActivity() {
053         Intent intent = new Intent();
054         intent.setClass(this, GameActivity.class);
055         //传递声音开关参数
056         intent.putExtra("audio", audio_on);
057         this.startActivity(intent);
058     }
059     //触摸屏单击事件
060     @Override
061     public boolean onTouchEvent(MotionEvent event) {
062             //如果当前在开场界面
063         if (currentView == introduceView) {
064             if (event.getAction() == MotionEvent.ACTION UP) {
065                 //进入帮助界面
066                 this.setContentView(helpView);
067                 this.currentView = helpView;
068             }
069             //如果当前在帮助界面
070         } else if (currentView == helpView) {
071             if (event.getAction() == MotionEvent.ACTION UP) {
072                 //进入声音设置界面
073                 this.setContentView(audioView);
074                 this.currentView = audioView;
075             }
076             //如果当前在声音设置界面
077         } else {
078             switch (event.getAction()) {
079             case MotionEvent.ACTION DOWN:
080                 //取得按下的 x、y 坐标
081                 float x = event.getX();
082                 float y = event.getY();
083                 //若单击位置在"是"上
084                 if (x > 30 && x < 60 && y > 177 && y < 207) {
085                     audioView.setClick(true);
086                     audioView.setAudio on(true);
087                     //更新界面
088                     audioView.invalidate();
089                     this.audio on = true;
090                 }
091                 //若单击位置在"否"上
092                 if (x > 330 && x < 360 && y > 177 && y < 207) {
093                     audioView.setClick(true);
094                     audioView.setAudio on(false);
095                     //更新界面
096                     audioView.invalidate();
097                     this.audio on = false;
098                 }
099                 break;
100             case MotionEvent.ACTION_UP:
101                 if (audioView.isClick()) {
102                     //进入游戏界面
103                     this.toGameActivity();
104                 }
105                 break;
106             }
107         }
108         return true;
```

```
109     }
110 }
```

（2）在 GameActivity 中先获得声音开关的变量值，接着初始化界面成全屏模式，最后设置界面显示 gameSurfaceView 并取得焦点。

```
01  package com.supermario.rabit;                    //声明包语句
02~06 行为引入相关类，这里不再列举，请阅读光盘内容
//……
07  //游戏主界面
08  public class GameActivity extends Activity {
09      private GameSurfaceView gameSurfaceView;
10      //声音默认为打开
11      private boolean audio_on = true;
12      @Override
13      protected void onCreate(Bundle savedInstanceState) {
14          // TODO Auto-generated method stub
15          super.onCreate(savedInstanceState);
16          //取得声音开关的参数
17          audio_on = this.getIntent().getBooleanExtra("audio", true);
18          gameSurfaceView = new GameSurfaceView(this);
19          //初始化界面
20          initWindow();
21          this.setContentView(gameSurfaceView);
22          //取得焦点
23          gameSurfaceView.requestFocus();
24      }
25      //设置全屏
26      public void initWindow(){
27          this.requestWindowFeature(Window.FEATURE NO TITLE);
28          this.getWindow().setFlags(WindowManager.LayoutParams.FLAG
            FULLSCREEN, WindowManager.LayoutParams.FLAG FULLSCREEN);
29      }
30      //若按下返回键则关闭界面
31      @Override
32      public boolean onKeyUp(int keyCode, KeyEvent event) {
33          gameSurfaceView.onKeyUp(keyCode, event);
34          if(keyCode == KeyEvent.KEYCODE_BACK){
35              this.finish();
36          }
37          return true;
38      }
39      //返回声音是否打开
40      public boolean isAudio on() {
41          return audio on;
42      }
43      //设置声音是否打开
44      public void setAudio_on(boolean audioOn) {
45          audio_on = audioOn;
46      }
47  }
```

（3）如下所示，为游戏运行主界面的实现代码，一开始声明游戏的角色类变量如小兔类、铃铛数组、背景视图类、音乐提供类等。整个初始化过程在构造函数中实现，在构造函数中初始化背景图片资源、初始化声音设置、显示最高分数和初始化铃铛列表。

游戏界面在 surfaceCreated()函数中将启动界面更新线程 refurbishThread 不断更新界

面，同时不断播放背景音乐。在线程 refurbishThread 的 run()函数中通过调用 update_all_components()更新当前界面所有元素的状态，再调用 onDraw()函数重新绘制界面，这样就产生了整个游戏连贯的效果。

游戏过程中需要不断处理 Touch 事件来更新小兔的目标 x 坐标，以此来控制小兔运动的方向。在界面销毁函数中，与界面创建函数对应的，需要停止刷新进程，并释放媒体播放器资源。

```
001 //游戏主界面
002 public class GameSurfaceView extends SurfaceView implements
003         SurfaceHolder.Callback {
004     //兔子
005     private Rabit rabit;
006     //刷新类
007     private RefurbishThread refurbishThread;
008     //铃铛
009     private List<Bell> bell_list = new ArrayList<Bell>();
010     //背景
011     private Backgroud bg;
012     //上下文
013     private Context context;
014     //铃铛制造类
015     private BellCreator bell_creator;
016     //小鸟
017     private Bird bird;
018     private SurfaceHolder holder;
019     private Bitmap bitmap_jump;
020     //向上跳按钮是否被按下
021     private boolean jump_button_clicked = false;
022     private boolean jump_comm_flag = false;
023     //是否播放音乐标志位
024     private boolean audio_on = false;
025     //音乐提供类
026     private AudioProvider audioProvider;
027     //分数
028     private int score = 0;
029     //最高分数
030     private int highest_score;
031     //撞击次数
032     private int hit_count = 0;
033     private Paint paint;
034     //游戏结束标志
035     private boolean game_over = false;
036     //结束界面
037     private Conclusion conclusion;
038     //构造函数
039     public GameSurfaceView(Context context) {
040         super(context);
041         this.context = context;
042         //实例化小兔
043         rabit = new Rabit(context);
044         //实例化背景
045         bg = new Backgroud(context);
046         //实例化结束界面
047         conclusion = new Conclusion(this);
048         bitmap_jump =
```

```
            BitmapFactory.decodeResource(context.getResources(),
049                 R.drawable.button);
050         //初始化声音设置
051         init_audio();
052         //初始化最高分数显示
053         init_highest_score();
054         //初始化铃铛制造类
055         bell creator = new BellCreator(context);
056         bird = new Bird(context);
057         //刷新类
058         refurbishThread = new RefurbishThread();
059         //初始化铃铛列表
060         init_bell_list();
061         paint = new Paint();
062         this.holder = this.getHolder();
063         holder.addCallback(this);
064         this.setFocusable(true);
065     }
066
067     private void init() {
068         rabit.init();
069         bg.init();
070         init_bird();
071         jump_button_clicked = false;
072         jump_comm_flag = false;
073         for (int i = 0; i < bell_list.size(); ++i) {
074             bell creator.recycle(bell list.get(i));
075         }
076         bell_creator.init();
077         init bell list();
078         //重新设置最高分数
079         highest score = (highest score > score) ? highest score : score;
080         score = 0;
081         hit count = 0;
082     }
083     //初始化小鸟
084     private void init bird() {
085         bird.setState(Bird.BIRD LEFT FLY0);
086         //设置小鸟初始位置
087         bird.setCenter x(240);
088         bird.setCenter y(30);
089         //显示在屏幕中
090         bird.setOn screen(true);
091     }
092     //初始化最高分显示
093     private void init highest score() {
094         SharedPreferences settings = ((GameActivity) context)
095                 .getPreferences(Activity.MODE PRIVATE);
096         //取得存储在文件中的最高纪录
097         highest score = settings.getInt("highestscore", 0);
098     }
099     //初始化音乐类
100     private void init audio() {
101         this.audio on = ((GameActivity) context).isAudio on();
102         if (audio on) {
103             audioProvider = new AudioProvider(context);
104         }
105     }
106     //初始化铃铛
```

```
107     private void init_bell_list() {
108         //制造出 5 个铃铛
109         Bell bell0 = bell_creator.createBell();
110         Bell bell1 = bell_creator.createBell();
111         Bell bell2 = bell_creator.createBell();
112         Bell bell3 = bell_creator.createBell();
113         Bell bell4 = bell_creator.createBell();
114         //设置铃铛的初始 Y 坐标
115         bell0.setCenter_y(170);
116         bell1.setCenter_y(130);
117         bell2.setCenter_y(90);
118         bell3.setCenter_y(50);
119         bell4.setCenter_y(10);
120         //将铃铛添加进铃铛列表中
121         bell_list.removeAll(bell_list);
122         bell_list.add(bell0);
123         bell_list.add(bell1);
124         bell_list.add(bell2);
125         bell_list.add(bell3);
126         bell_list.add(bell4);
127     }
128     //绘制界面
129     protected void onDraw(Canvas canvas) {
130         super.onDraw(canvas);
131         bg.onDraw(canvas);
132         //如果为单击跳跃按钮，则在右下角显示跳跃按钮
133         if (!jump_button_clicked) {
134             canvas.drawBitmap(bitmap_jump, 333, 199, paint);
135         }
136         //绘制铃铛
137         drawBell(canvas);
138         //绘制小兔
139         rabit.onDraw(canvas);
140         //绘制分数
141         drawScore(canvas);
142         //如果游戏结束，则绘制结束界面
143         if (game_over) {
144             conclusion.onDraw(canvas);
145         }
146         //如果小鸟在屏幕则绘制小鸟
147         if (bird.isOn_screen()) {
148             bird.onDraw(canvas);
149         }
150     }
151     //绘制铃铛
152     private void drawBell(Canvas canvas) {
153         for (int i = 0; i < bell_list.size(); ++i) {
154             bell_list.get(i).onDraw(canvas);
155         }
156     }
157     //绘制分数
158     private void drawScore(Canvas canvas) {
159         //设置字体颜色为白色
160         paint.setColor(Color.WHITE);
161         paint.setTextSize(15);
162         canvas.drawText("" + score, 22, 22, paint);
163     }
164     //监听按键
165     public boolean onKeyUp(int keyCode, KeyEvent event) {
```

```
166         if(keyCode == KeyEvent.KEYCODE BACK) {
167             this.refurbishThread.setGo on(false);
168         }
169         return true;
170     }
171     //监听触摸事件
172     public boolean onTouchEvent(MotionEvent event) {
173         //获得触摸位置的 x、y 坐标
174         float x = event.getX();
175         float y = event.getY();
176         //如果游戏已经结束
177         if (game_over) {
178             //游戏结束后的触屏处理
179             switch (event.getAction()) {
180             //单击了按钮区域
181             case MotionEvent.ACTION DOWN:
182                 if (x >= 165 && x <= 237 && y >= 154 && y <= 180) {
183                     conclusion.setPressed(true);
184                 }
185                 break;
186             case MotionEvent.ACTION_MOVE:
187                 break;
188             //单击了非按钮区域
189             case MotionEvent.ACTION UP:
190                 if (x >= 165 && x <= 237 && y >= 154 && y <= 180) {
191                     conclusion.setPressed(false);
192                     init();
193                     game over = false;
194                 }
195                 break;
196             default:
197                 break;
198             }
199         } else {
200             //游戏中的触屏处理
201             if (jump_button_clicked) {
202                 //设置小兔的 x 轴目标位置
203                 switch (event.getAction()) {
204                 case MotionEvent.ACTION DOWN:
205                 case MotionEvent.ACTION MOVE:
206                 case MotionEvent.ACTION UP:
207                     rabit.setX_destination(x);
208                     break;
209                 default:
210                     rabit.setX_destination(rabit.getX());
211                 }
212             } else {
213                 //单点击了跳跃按钮
214                 if (x > 327 && x < 364 && y > 196 && y < 230) {
215                     if(event.getAction() == MotionEvent.ACTION UP){
216                         jump comm flag = true;
217                         jump button clicked = true;
218                     }
219                 } else {
220                     switch (event.getAction()) {
221                     case MotionEvent.ACTION DOWN:
222                     case MotionEvent.ACTION MOVE:
223                     case MotionEvent.ACTION_UP:
224                         rabit.setX_destination(x);
225                         break;
```

```
226                 default:
227                     rabit.setX destination(rabit.getX());
228                 }
229             }
230         }
231     }
232     return true;
233 }
234 //界面改变
235 public void surfaceChanged(SurfaceHolder holder, int format, int
    width,
236         int height) {
237 }
238 //界面初始化
239 public void surfaceCreated(SurfaceHolder holder) {
240     refurbishThread.setGo_on(true);
241     //启动线程
242     refurbishThread.start();
243     if (audio on) {
244         //播放背景音乐
245         audioProvider.play_bg();
246     }
247 }
248 //界面销毁
249 @Override
250 public void surfaceDestroyed(SurfaceHolder holder) {
251     // TODO Auto-generated method stub
252     refurbishThread.setGo_on(false);
253     if (audio_on) {
254         //释放媒体资源
255         audioProvider.release();
256     }
257     //保存最高分
258     highest score = (highest score > score) ? highest score : score;
259     SharedPreferences pre = ((GameActivity) context).
        getPreferences(0);
260     SharedPreferences.Editor editor = pre.edit();
261     editor.putInt("highestscore", highest_score);
262 }
263 //刷新进程
264 class RefurbishThread extends Thread {
265     private boolean go_on = false;
266     //取得是否继续
267     public boolean isGo_on() {
268         return go on;
269     }
270     //设置是否继续
271     public void setGo on(boolean goOn) {
272         go on = goOn;
273     }
274     //运行
275     public void run() {
276         while (go on) {
277             try {
278                 // Thread.sleep(50);
279                 Thread.sleep(100);
280             } catch (InterruptedException e) {
281                 // TODO Auto-generated catch block
282                 e.printStackTrace();
```

```
283                 }
284                 //更新所有组件
285                 update_all_components();
286                 synchronized (holder) {
287                     Canvas canvas = holder.lockCanvas();
288                     //绘制画面
289                     GameSurfaceView.this.onDraw(canvas);
290                     holder.unlockCanvasAndPost(canvas);
291                 }
292             }
293         }
294     }
```

（4）前面提到了更新界面所有组件的函数 update_all_components()，该函数中主要执行 3 个函数：update_rabit()、update_bells()、update_bird()来分别更新小兔、铃铛和小鸟的状态，如以下代码 287～297 行所示。

在 update_rabit()函数中首先判断小兔是否在地面，若在地面则根据触摸点与小兔的方位关系决定小兔的运动目的地，并显示奔跑动画。若小兔在空中，则不仅要处理小兔水平方向的运行，还要处理垂直方向的运动，水平方向的运动与地面类似，垂直方向的运动分为 6 个阶段，小兔先是上升，到最后一阶段之后转而进入下降状态。

处理小兔上升和下降的函数分别为 rabit_move_up()和 rabit_move_down()，根据小兔在屏幕中的位置采用分段处理的方式。在小兔上升时，若小兔位置在屏幕中心线 1/2 以下处则只更新小兔的坐标，若小兔位置在屏幕 2/3～1/2 处，则小兔和铃铛、背景、小鸟分别以小兔的半速相对运动，若小兔超过屏幕 2/3 则小兔不移动，其他背景元素全速下移。在上升和下降函数中要及时移除出现在视野之外的铃铛。

在 update_bells()函数中，首先判断铃铛的状态，若处于爆炸状态则显示爆炸动画。否则判断小兔是否与铃铛碰撞，若有碰撞则将铃铛状态设置为爆炸，并更新分数。

在 update_bird()函数中，若满足一定条件则在屏幕中显示小鸟，小鸟被击中后将加速飞行同时播放鸟叫声音，分数翻一倍。

```
001 //更新小兔的状态
002 private void update rabit(){
003     //如果小兔在地面
004     if (rabit.isOnGround()) {
005         //处理水平方向移动
006         // rabit 到达目的位置
007         if (Math.abs(rabit.getX() - rabit.getX destination()) < 5) {
008             //如果小兔面朝左
009             if (rabit.isFaceLeft()) {
010                 //小兔状态为面朝左停下
011                 rabit.setPic state(Rabit.RABIT PIC LEFT STOP);
012                 //设置小兔朝向
013                 rabit.setFace_state(Rabit.RABIT_FACE_LEFT);
014                 //小兔在地面状态为停下
015                 rabit.setGround state(Rabit.RABIT LEFT STOP);
016             } else {
017                 //小兔状态为面朝右停下
018                 rabit.setPic state(Rabit.RABIT PIC RIGHT STOP);
019                 //设置小兔朝向
020                 rabit.setFace_state(Rabit.RABIT_FACE_RIGHT);
021                 //小兔在地面状态为停下
```

```
022                 rabit.setGround state(Rabit.RABIT RIGHT STOP);
023             }
024         } else if (rabit.getX() - rabit.getX destination() >= 5) {
025             // rabit 在 destination 的右面，它将向左移动
026             rabit.setFace_state(Rabit.RABIT_FACE_LEFT);
027             rabit.setX(rabit.getX() - Rabit.SPEED_X);
028                 //小兔在地面上向左运动显示的图片 0
029             if (rabit.getGround state() ==
            Rabit.RABIT LEFT MOVE1 ON GROUND) {
030                 rabit.setGround state(Rabit.RABIT LEFT
                 MOVE2 ON GROUND);
031                 rabit.setPic state(Rabit.RABIT PIC ON GROUND
                 LEFT JUMP1);
032                 //小兔在地面上向左运动显示的图片 1
033             } else if (rabit.getGround_state() ==
             Rabit.RABIT_LEFT_MOVE2_ON_GROUND) {
034                 rabit.setGround_state(Rabit.RABIT_LEFT_MOVE1
                 _ON_GROUND);
035                 rabit.setPic state(Rabit.RABIT PIC ON GROUND
                 LEFT JUMP0);
036                 //小兔在地面向左运动一开始显示的图片
037             } else {
038               rabit.setGround state(Rabit.RABIT LEFT
               MOVE1 ON GROUND);
039                 rabit.setPic state(Rabit.RABIT PIC ON GROUND
                 LEFT JUMP0);
040             }
041
042         } else if (rabit.getX_destination() - rabit.getX() > 5) {
043             // rabit 在 destination 的左面，它将向右移动
044             rabit.setX(rabit.getX() + Rabit.SPEED X);
045             rabit.setFace state(Rabit.RABIT FACE RIGHT);
046                 //小兔在地面上向右运动显示的图片 0
047             if (rabit.getGround state() ==
             Rabit.RABIT_RIGHT_MOVE1_ON_GROUND) {
048                 rabit.setGround_state(Rabit.RABIT_RIGHT_MOVE2
                 _ON_GROUND);
049                 rabit.setPic_state(Rabit.RABIT_PIC_ON_GROUND_RIGHT
                 _JUMP1);
050                 //小兔在地面上向右运动显示的图片 1
051             } else if (rabit.getGround state() ==
            Rabit.RABIT RIGHT MOVE2 ON GROUND) {
052                 rabit.setGround state(Rabit.RABIT RIGHT MOVE1
                 ON GROUND);
053                 rabit.setPic state(Rabit.RABIT PIC ON
                 GROUND RIGHT JUMP0);
054             } else {
055                 //小兔在地面向右运动一开始显示的图片
056                 rabit.setGround_state(Rabit.RABIT_RIGHT_MOVE1_ON_
                 GROUND);
057                 rabit.setPic state(Rabit.RABIT PIC ON GROUND
                 RIGHT JUMP0);
058             }
059         }
060         //单击了跳跃按钮
061         if (jump comm flag) {
062             rabit.setAir state(Rabit.RABIT ON AIR UP0);
063             rabit.setGround_state(Rabit.RABIT_NOT_ON_GROUND);
064             rabit.setY(rabit.getY() - Rabit.SPEED_Y);
065             //面朝左
```

```
066             if (rabit.isFaceLeft()) {
067                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_LEFT_JUMP);
068             } else {
069                 //面朝右
070                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_RIGHT_JUMP);
071             }
072             jump_comm_flag = false;
073         }
074
075     } else {
076         // rabit 在空中
077         //处理水方向移动
078         if (rabit.getX() - rabit.getX_destination() >= 10) {
079             // rabit 在 destination 的右面，它将向左移动
080             rabit.setFace_state(Rabit.RABIT_FACE_LEFT);
081             rabit.setX(rabit.getX() - Rabit.SPEED_X_ON_AIR);
082         } else if (rabit.getX_destination() - rabit.getX() > 10) {
083             // rabit 在 destination 的左面，它将向右移动
084             rabit.setX(rabit.getX() + Rabit.SPEED_X_ON_AIR);
085             rabit.setFace_state(Rabit.RABIT_FACE_RIGHT);
086         }
087         //处理垂直方向移动
088         if (rabit.getAir_state() == Rabit.RABIT_ON_AIR_UP0) {
089             //小兔向上运动
090             rabit_move_up();
091             rabit.setAir_state(Rabit.RABIT_ON_AIR_UP1);
092             //小兔朝左跳
093             if (rabit.isFaceLeft())
094                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_LEFT_JUMP);
095             //小兔朝右跳
096             else
097                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_RIGHT_JUMP);
098             //上升状态 1
099         } else if (rabit.getAir_state() == Rabit.RABIT_ON_AIR_UP1) {
100             // rabit.setY(rabit.getY()-Rabit.SPEED_Y);
101             rabit_move_up();
102             rabit.setAir_state(Rabit.RABIT_ON_AIR_UP2);
103             if (rabit.isFaceLeft())
104                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_LEFT_JUMP);
105             else
106                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_RIGHT_JUMP);
107             //上升状态 2
108         } else if (rabit.getAir_state() == Rabit.RABIT_ON_AIR_UP2) {
109             // rabit.setY(rabit.getY()-Rabit.SPEED_Y);
110             rabit_move_up();
111             rabit.setAir_state(Rabit.RABIT_ON_AIR_UP3);
112             if (rabit.isFaceLeft())
113                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_LEFT_JUMP);
114             else
115                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_RIGHT_JUMP);
116             //上升状态 3
117         } else if (rabit.getAir_state() == Rabit.RABIT_ON_AIR_UP3) {
118             // rabit.setY(rabit.getY()-Rabit.SPEED_Y);
119             rabit_move_up();
120             rabit.setAir_state(Rabit.RABIT_ON_AIR_UP4);
121             if (rabit.isFaceLeft())
122                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_LEFT_JUMP);
123             else
124                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_RIGHT_JUMP);
```

```
125             //上升状态 4
126         } else if (rabit.getAir_state() == Rabit.RABIT_ON_AIR_UP4) {
127             // rabit.setY(rabit.getY()-Rabit.SPEED_Y);
128             rabit_move_up();
129             rabit.setAir_state(Rabit.RABIT_ON_AIR_UP5);
130             if (rabit.isFaceLeft())
131                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_LEFT_JUMP);
132             else
133                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_RIGHT_JUMP);
134             //上升状态 5
135         } else if (rabit.getAir_state() == Rabit.RABIT_ON_AIR_UP5) {
136             rabit.setAir_state(Rabit.RABIT_ON_AIR_STOP);
137             if (rabit.isFaceLeft())
138                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_LEFT_STOP);
139             else
140                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_RIGHT_STOP);
141             //停止上升
142         } else if (rabit.getAir_state() == Rabit.RABIT_ON_AIR_STOP) {
143             rabit.setAir_state(Rabit.RABIT_ON_AIR_DOWN);
144             if (rabit.isFaceLeft())
145                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_LEFT_STOP);
146             else
147                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_RIGHT_STOP);
148             //开始往下掉
149         } else if (rabit.getAir_state() == Rabit.RABIT_ON_AIR_DOWN) {
150             // rabit.setY(rabit.getY() + Rabit.SPEED_Y);
151             rabit.setAir_state(Rabit.RABIT_ON_AIR_DOWN);
152             if (rabit.isFaceLeft())
153                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_LEFT_DOWN);
154             else
155                 rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR_RIGHT_DOWN);
156             rabit_move_down();
157         }
158     }
159 }
160 //更新铃铛的状态
161 private void update_bells(){
162     for (int i = 0; i < bell_list.size(); ++i) {
163         Bell bell = bell_list.get(i);
164         //铃铛爆炸
165         if (bell.isExplode()) {
166             //爆炸状态 0
167             if (bell.getState() == Bell.BELL_EXPLODE0) {
168                 bell.setState(Bell.BELL_EXPLODE1);
169             //爆炸状态 1
170             } else if (bell.getState() == Bell.BELL_EXPLODE1) {
171                 bell.setState(Bell.BELL_EXPLODE2);
172             //爆炸状态 2
173             } else if (bell.getState() == Bell.BELL_EXPLODE2) {
174                 bell_list.remove(bell);
175                 bell_creator.recycle(bell);
176                 --i;
177             }
178         } else {
179             if (rabit.isHitBell(bell)) {
180                 ++hit_count;
181                 //分数累加
182                 score += hit_count * 10;
183                 //播放铃铛声音
```

```
184                 if(audio on) audioProvider.play bell ding();
185                 bell.setState(Bell.BELL EXPLODE0);
186                 rabit.setAir state(Rabit.RABIT ON AIR UP0);
187                 if (rabit.isFaceLeft()) {
188                     rabit.setPic state
                        (Rabit.RABIT PIC ON AIR LEFT JUMP);
189                 } else {
190                     rabit.setPic state(Rabit.RABIT PIC ON AIR RIGHT
                         JUMP);
191                 }
192             }
193         }
194     }
195 }
196 //更新小鸟的状态
197 private void update bird(){
198     if (bird.isOn screen()) {
199         //处理垂直
200         if (bird.getCenter y() > Constant.SCREEN HEIGHT) {
201             bird.setOn_screen(false);
202         }
203     }
204     //若小鸟在屏幕
205     if (bird.isOn screen()) {
206         //小鸟被击中
207         if (bird.isHited()) {
208             if (bird.isFaceLeft()) {
209                 //加速
210                 bird.setCenter_x(bird.getCenter_x() - Bird.BIRD_SPEED_
                    POWER);
211                 if (bird.getCenter x() < -5) {
212                     bird.setOn screen(false);
213                 }
214             } else {
215                 //加速
216                 bird.setCenter_x(bird.getCenter_x()
217                             + Bird.BIRD_SPEED_POWER);
218                 if (bird.getCenter_x() > Constant.SCREEN_WIDTH + 5) {
219                     bird.setOn_screen(false);
220                 }
221             }
222         } else {
223             if (bird.isFaceLeft()) {
224                 //正常速度
225                 bird.setCenter x(bird.getCenter x() - Bird.BIRD SPEED);
226                 if (bird.getCenter x() < -5) {
227                     bird.setCenter x(0);
228                     bird.setState(Bird.BIRD RIGHT FLY0);
229                 } else {
230                     if (rabit.isHitBird(bird)) {
231                         ++hit_count;
232                         //分数翻倍
233                         score *= 2;
234                         //播放鸟叫
235                         if(audio_on) audioProvider.play_twitter();
236                         bird.setState(Bird.BIRD_LEFT_FLY_POWER);
237                         rabit.setAir_state(Rabit.RABIT_ON_AIR_UP0);
238                         if (rabit.isFaceLeft()) {
239                             rabit.setPic_state(Rabit.RABIT_PIC_ON_AIR
                            _LEFT_JUMP);
```

```
240                     } else {
241                         rabit.setPic state(Rabit.RABIT PIC ON AIR
                             RIGHT JUMP);
242                     }
243                 } else if (bird.getState() == Bird.BIRD LEFT FLY0) {
244                     bird.setState(Bird.BIRD LEFT FLY1);
245                 } else if (bird.getState() == Bird.BIRD LEFT FLY1) {
246                     bird.setState(Bird.BIRD LEFT FLY2);
247                 } else {
248                     bird.setState(Bird.BIRD LEFT FLY0);
249                 }
250             }
251         } else {
252             bird.setCenter_x(bird.getCenter_x() + Bird.BIRD_SPEED);
253             if (bird.getCenter_x() > Constant.SCREEN_WIDTH + 5) {
254                 bird.setCenter_x(Constant.SCREEN_WIDTH);
255                 bird.setState(Bird.BIRD_LEFT_FLY0);
256             } else {
257                 if (rabit.isHitBird(bird)) {
258                     ++hit count;
259                     score *= 2;
260                     if(audio on) audioProvider.play twitter();
261                     bird.setState(Bird.BIRD LEFT FLY POWER);
262                     rabit.setAir state(Rabit.RABIT ON AIR UP0);
263                     if (rabit.isFaceLeft()) {
264                         rabit
265                                 .setPic state(Rabit.RABIT PIC ON AIR L
                                EFT JUMP);
266                     } else {
267                         rabit
268                                 .setPic state(Rabit.RABIT PIC ON AIR R
                                IGHT JUMP);
269                     }
270                 } else if (bird.getState() == Bird.BIRD_RIGHT_FLY0) {
271                     bird.setState(Bird.BIRD_RIGHT_FLY1);
272                 } else if (bird.getState() == Bird.BIRD_RIGHT_FLY1) {
273                     bird.setState(Bird.BIRD_RIGHT_FLY2);
274                 } else {
275                     bird.setState(Bird.BIRD_RIGHT_FLY0);
276                 }
277             }
278         }
279     }
280 }
281 //添加一个 bird 的逻辑
282 if (hit count % 11 == 0 && bird.isOn screen() == false
283         && rabit.getY() < 210) {
284     init bird();
285 }
286 }
287 //更新所有组件
288 private void update all components() {
289     if (game over)
290         return;
291     //更新小兔
292     update rabit();
293     //更新铃铛
294     update_bells();
295     //更新小鸟
296     update_bird();
```

```
297 }
298 //小兔向上运动
299 private void rabit move up() {
300     //如果小兔的位置在屏幕中线以下时，只更新小兔的坐标
301     if (rabit.getCenter_y() > Constant.SCREEN_HEIGHT / 2) {
302         rabit.setY(rabit.getY() - Rabit.SPEED_Y);
303         return;
304     }
305     //处理屏幕背景的变化
306     //处理 bells 的位置
307     //所有的 bell 下移
308     //rabit 跳跃至屏幕 1/2 处~2/3 时，处理整个场景 Components 变化
309     if (rabit.getCenter_y() > 0.33 * Constant.SCREEN_HEIGHT) {
310         for (int i = 0; i < bell_list.size(); ++i) {
311             Bell bell = bell_list.get(i);
312             //所有铃铛的位置下移
313             bell.setCenter y(bell.getCenter y() + Rabit.SPEED Y / 2);
314         }
315         //背景以半速下移
316         bg.drag_down((int) Rabit.SPEED_Y / 2);
317         //小鸟以半速下移
318         bird.setCenter y(bird.getCenter y() + Rabit.SPEED Y / 2);
319         //小兔以半速上移
320         rabit.setCenter_y(rabit.getCenter_y() - Rabit.SPEED_Y / 2);
321     } else {
322         // rabit 跳跃比屏幕的 2/3 处还高时，处理整个场景的变化
323         for (int i = 0; i < bell list.size(); ++i) {
324             Bell bell = bell list.get(i);
325             //铃铛以全速下移
326             bell.setCenter y(bell.getCenter y() + Rabit.SPEED Y);
327         }
328         bg.drag_down((int) Rabit.SPEED_Y);
329         if (bird.isOn_screen()) {
330             //小鸟以全速下移
331             bird.setCenter y(bird.getCenter y() + Rabit.SPEED Y);
332         }
333     }
334     //是否需要清理落地的 bell， bell_list[0] 处于最底处，判断它是否落地
335     Bell lowBell = bell_list.get(0);
336     if (lowBell != null) {
337         if (lowBell.getCenter_y() > 230) {
338             bell_list.remove(lowBell);
339             //清理消失在屏幕视野的铃铛
340             bell creator.recycle(lowBell);
341         }
342     }
343     //判断是否需要添加新的 bell， bell_list[size-1] 处于最高处，通过它来判断
344     if (bell_list.size() > 0) {
345         Bell upBell = bell_list.get(bell_list.size() - 1);
346         if (upBell.getCenter_y() > 50)
347             bell_list.add(bell_creator.createBell());
348     }
349 
350 }
351 //小兔向下移动
352 private void rabit move down() {
353     if (rabit.getY() > 210) {
354         //rabit 已经下落至屏幕底部，不可能再碰到 bell、rabit 高空下坠时背景，
```

```
            bell,bird 的位置处理
355         if (bg.isLowest()) {
356             game over = true;
357             rabit.setY(Constant.RABIT_INIT_Y);
358             if (rabit.isFaceLeft()) {
359                 rabit.setPic_state(Rabit.RABIT_PIC_LEFT_STOP);
360             } else {
361                 rabit.setPic_state(Rabit.RABIT_PIC_RIGHT_STOP);
362             }
363             bg.init();
364         } else {
365             //背景向上拖动
366             bg.drag up((int) Rabit.SPEED Y);
367             //屏幕上的 bird 向上移动
368             if (bird.isOn_screen()) {
369                 bird.setCenter_y(bird.getCenter_y() - Rabit.SPEED_Y);
370                 if (bird.getCenter_y() < 0) {
371                     bird.setOn_screen(false);
372                 }
373             }
374             //所有的 bell 向上移动
375             for (int i = 0; i < bell_list.size(); ++i) {
376                 Bell bell = bell_list.get(i);
377                 bell.setCenter y(bell.getCenter y() - Rabit.SPEED Y);
378             }
379             //处理移动到屏幕外的 bell
380             if (bell list.size() > 0) {
381                 Bell bell up = bell list.get(bell list.size() - 1);
382                 if (bell up.getCenter y() < -Bell.BELL OK HEIGHT / 2) {
383                     //移除铃铛
384                     bell list.remove(bell up);
385                     bell creator.recycle(bell up);
386                 }
387             }
388         }
389     } else {
390         // rabit 还没有玩完
391         rabit.setY(rabit.getY() + Rabit.SPEED Y);
392     }
393 }
394 //取得分数
395 public int getScore() {
396     return score;
397 }
398 //设置分数
399 public void setScore(int score) {
400     this.score = score;
401 }
402 //取得最高分
403 public int getHighest_score() {
404     return highest_score;
405 }
406 //设置最高分
407 public void setHighest_score(int highestScore) {
408     highest score = highestScore;
409 }
```

20.6　知 识 拓 展

本章在处理游戏声音的时候使用了 SoundPool，它一般用于播放一些短的反应速度要求高的声音，如游戏中的爆破声。而 MediaPlayer 适合播放长点的。SoundPool 有如下特点：

（1）SoundPool 载入音乐文件使用了独立的线程，不会阻塞 UI 主线程的操作。但是如果音效文件过大没有载入完毕，我们调用 play 方法时可能产生严重的后果。这里 Android SDK 提供了一个 SoundPool.OnLoadCompleteListener 类来帮助我们了解媒体文件是否载入完毕，重载 onLoadComplete(SoundPool soundPool, int sampleId, int status) 方法即可获得。

（2）从上面的 onLoadComplete 方法可以看出该类有很多参数，比如类似 id，使 SoundPool 在 load 时可以处理多个媒体一次初始化并放入内存中，效率比 MediaPlayer 高了很多。

（3）SoundPool 类支持同时播放多个音效，这对于游戏来说是十分必要的。而 MediaPlayer 类是同步执行的，只能一个文件一个文件地播放。

20.7　本 章 小 结

本章介绍了小兔跳铃铛游戏的开发过程。本章的游戏要处理的动画有很多，这需要读者对 Android 绘图类有充分的了解。本章的难点在于小兔运动过程中一系列逻辑的判断，包括小兔的运动、背景的移动等，需要读者具有一定的空间想象力和数学运算能力。

第 21 章　飞行射击游戏

在众多游戏类型中，飞行射击游戏是一种很典型的游戏，这种类型的游戏操作简单、画面炫丽。本章要介绍的就是如何开发飞行射击游戏，通过一个简单的例子讲解射击游戏射击的要领。

21.1　功 能 分 析

我们先来看一下游戏的效果图，这样对整个游戏先有个形象的了解。如图 21.1 所示，是游戏的一个画面截图。

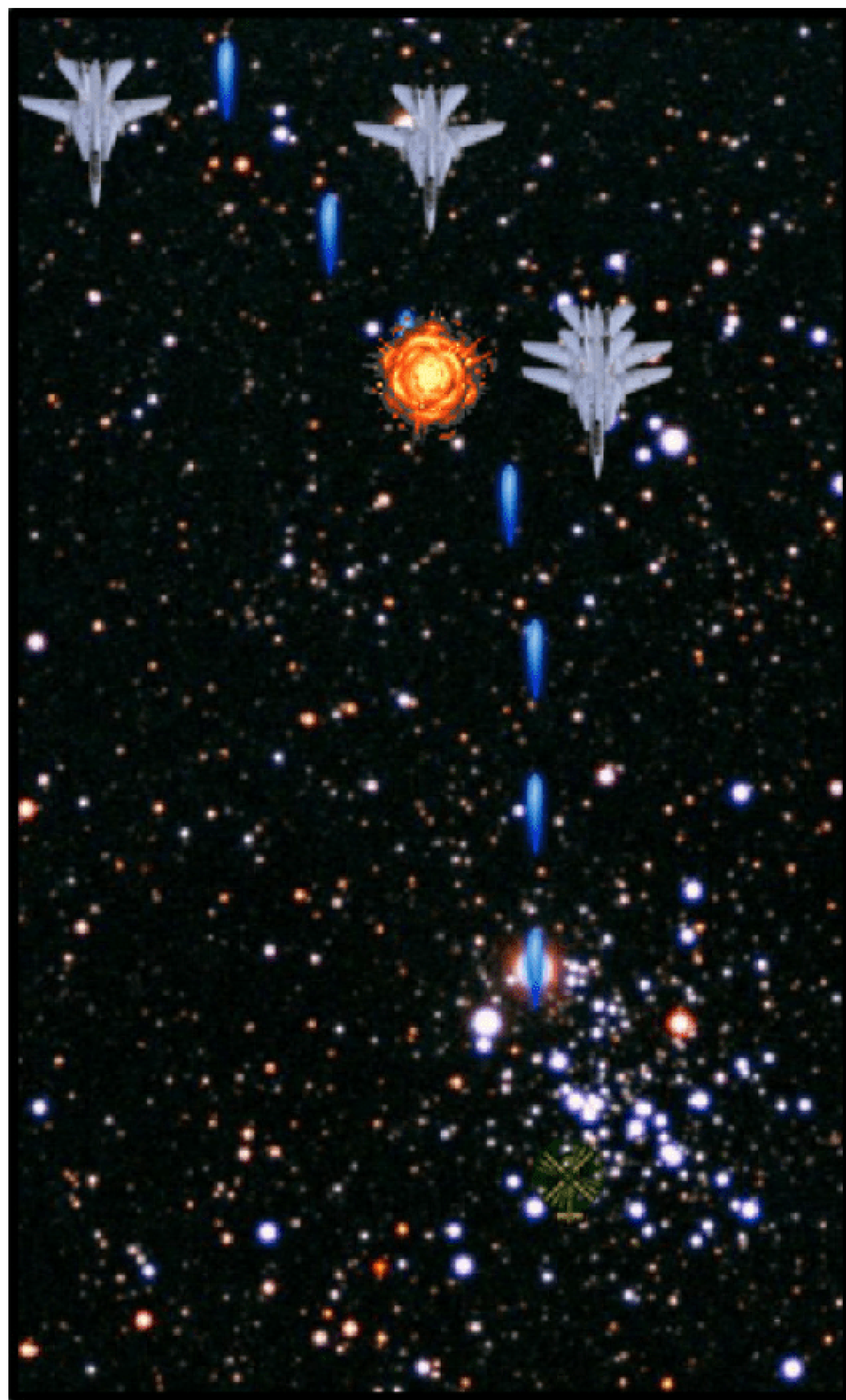


图 21.1　游戏画面截图

从图中我们可以看到游戏的组成有主角、敌机、子弹、背景地图，基本上游戏开发就

围绕着这 4 样元素进行。以前大家玩飞行射击游戏，应该都有这种感觉，就是飞机不断地前进，场景不断更换，这是如何实现的呢？其实这是利用相对运动的原理，如图 21.2 所示，将背景地图不断往下拉，这就制造了当前飞机不断前进的“假象”。当地图的顶端翻滚到底部的时候，将顶部的坐标置为-Height，其中 Height 是背景图片的高度。

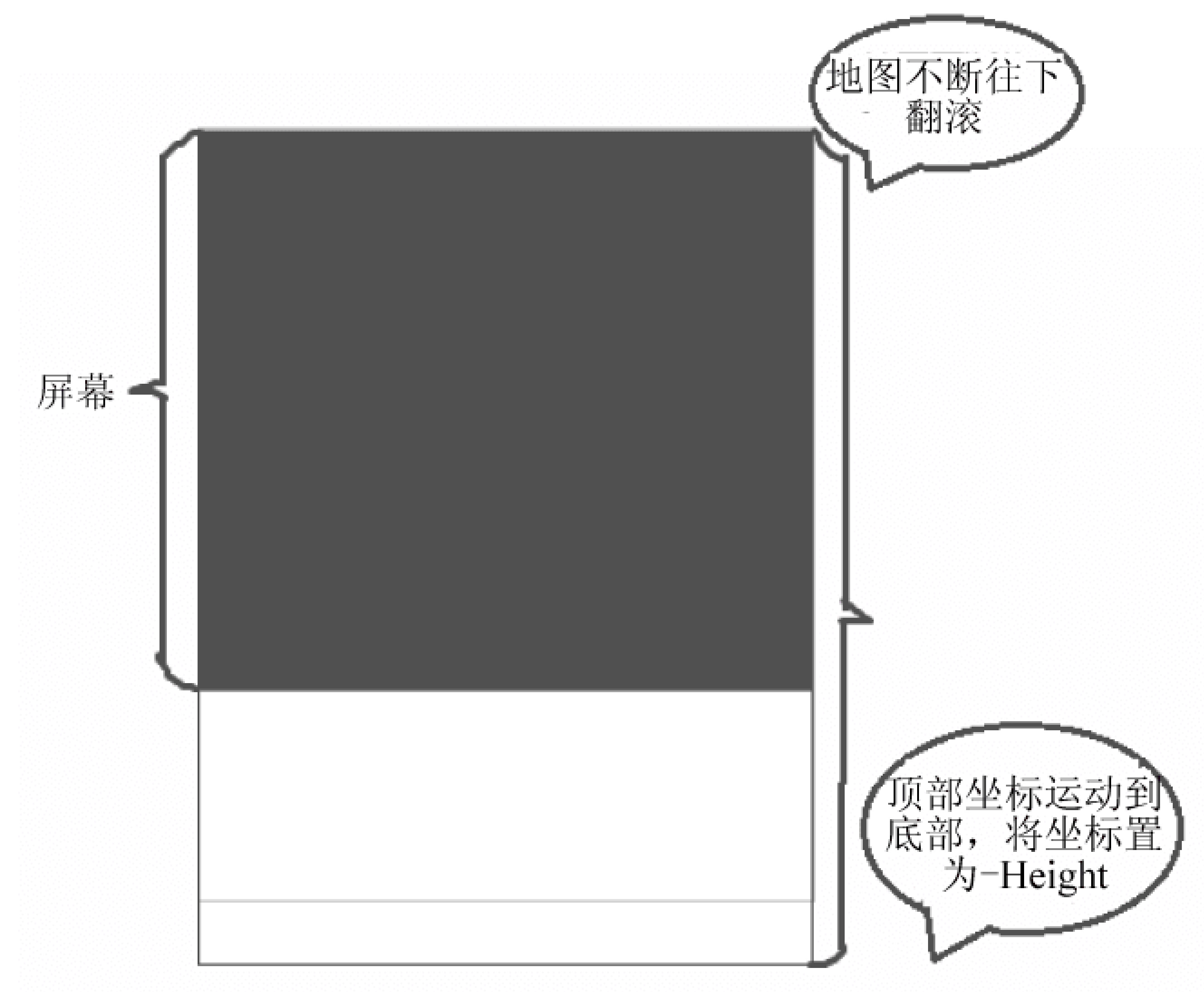


图 21.2　背景地图实现原理

游戏过程中还要处理与玩家的互动，包括飞机的移动和子弹击中敌机的处理。当用户单击触摸屏时，将触发 onTouchEvent 函数，我们需要在这个函数中实现飞机的移动。子弹与敌机的碰撞可以理解为两个矩形的碰撞，进一步，可以将这个子弹的矩形简化成一个点，判断这个点与矩形的交集情况，进而判断出子弹是否与敌机碰撞。

21.2　子弹和敌机类的实现

游戏的过程其实就是由一个个动画拼凑而成，因此首先我们要实现一个能播放动画的类 Animation。如下所示，实现了一个简单的动画播放类。大家知道由于人的视觉残留效应，如果低于某个时间间隔连续播放几幅静态图片，人眼将分辨不出画面的切换过程，因此我们动画的实现原理也一样，在低于某个时间间隔内不断绘制图片。如代码 73～99 行所示，先判断结束播放标志位 mIsend，如果为 false 则继续播放。接着比较当前时间和上一帧画面的绘制时间，如果超过指定时间则将帧 ID 值加 1，并更新上一帧绘制时间为当前时间。当播放到最后一帧，则判断循环标志位，如果设置了循环播放，则将帧 ID 值重置，否则设置结束标志位为 true。

```
001 package guo.supermario.shooting;                    //声明包语句
002~009 行为引入相关类，这里不再列举，请阅读光盘内容
//……
010 public class Animation {
```

```
011     /** 上一帧播放时间 **/
012     private long mLastPlayTime = 0;
013     /** 播放当前帧的 ID **/
014     private int mPlayID = 0;
015     /** 动画 frame 数量 **/
016     private int mFrameCount = 0;
017     /** 用于存储动画资源图片 **/
018     private Bitmap[] mframeBitmap = null;
019     /** 是否循环播放 **/
020     private boolean mIsLoop = false;
021     /** 播放结束 **/
022     public boolean mIsend = false;
023     /** 动画播放间隙时间 **/
024     private static final int ANIM TIME = 30;
025
026     /**
027      * 构造函数
028      * @param context
029      * @param frameBitmapID
030      * @param isloop
031      */
032     public Animation(Context context, int [] frameBitmapID, boolean
        isloop) {
033     mFrameCount = frameBitmapID.length;
034     mframeBitmap = new Bitmap[mFrameCount];
035     for(int i =0; i < mFrameCount; i++) {
036        mframeBitmap[i] = ReadBitMap(context,frameBitmapID[i]);
037     }
038     mIsLoop = isloop;
039     }
040
041     /**
042      * 构造函数
043      * @param context
044      * @param frameBitmap
045      * @param isloop
046      */
047     public Animation(Context context, Bitmap [] frameBitmap, boolean
        isloop) {
048     mFrameCount = frameBitmap.length;
049     mframeBitmap = frameBitmap;
050     mIsLoop = isloop;
051     }
052
053     //重置动画
054     public void reset() {
055     mLastPlayTime = 0;
056     mPlayID =0;
057     mIsend= false;
058     }
059
060     /**
061     * 绘制动画中的其中一帧
062     * @param Canvas
063     * @param paint
064     * @param x
065     * @param y
066     * @param frameID
```

```
067     */
068     public void DrawFrame(Canvas Canvas, Paint paint, int x, int y,int
        frameID) {
069         Canvas.drawBitmap(mframeBitmap[frameID], x, y, paint);
070     }
071
072
073     /**
074      * 绘制动画
075      * @param Canvas
076      * @param paint
077      * @param x
078      * @param y
079      */
080     public void DrawAnimation(Canvas Canvas, Paint paint, int x, int y){
081     //如果没有播放结束则继续播放
082     if (!mIsend) {
083         Canvas.drawBitmap(mframeBitmap[mPlayID], x, y, paint);
084         long time = System.currentTimeMillis();
085         if (time - mLastPlayTime > ANIM_TIME) {
086             mPlayID++;
087             mLastPlayTime = time;
088             if (mPlayID >= mFrameCount) {
089                 //标志动画播放结束
090                 mIsend = true;
091                 if (mIsLoop) {
092                     //设置循环播放
093                     mIsend = false;
094                     mPlayID = 0;
095                 }
096             }
097         }
098     }
099     }
100
101    /**
102      * 读取图片资源
103      * @param context
104      * @param resId
105      * @return
106      */
107     public Bitmap ReadBitMap(Context context, int resId) {
108     BitmapFactory.Options opt = new BitmapFactory.Options();
109     opt.inPreferredConfig = Bitmap.Config.RGB_565;
110     opt.inPurgeable = true;
111     opt.inInputShareable = true;
112     // 获取资源图片
113     InputStream is = context.getResources().openRawResource(resId);
114     return BitmapFactory.decodeStream(is, null, opt);
115     }
```

新建子弹类 Bullet，实现代码如下所示，在构造函数中实例化一个 Animation 类，并采用 Animation 的 DrawAnimation 绘制动画。

```
01  package guo.supermario.shooting;                    //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容
//……
08  public class Bullet {
```

```
09      /**子弹的 X 轴速度**/
10      static final int BULLET_STEP_X = 3;
11      /**子弹的 Y 轴速度**/
12      static final int BULLET_STEP_Y = 15;
13      /**子弹图片的宽度**/
14      static final int BULLET_WIDTH = 40;
15      /** 子弹的 X、Y 坐标 **/
16      public int m_posX = 0;
17      public int m_posY = 0;
18      /** 子弹的动画 **/
19      private Animation mAnimation = null;
20      /**是否更新绘制子弹**/
21      boolean mFacus = false;
22      Context mContext = null;
23      public Bullet(Context context, Bitmap[] frameBitmap) {
24          mContext = context;
25          mAnimation = new Animation(mContext, frameBitmap, true);
26      }
27      /**初始化坐标**/
28      public void init(int x, int y) {
29          m_posX = x;
30          m_posY = y;
31          mFacus = true;
32      }
33      /**绘制子弹**/
34      public void DrawBullet(Canvas Canvas, Paint paint) {
35          if (mFacus) {
36              mAnimation.DrawAnimation(Canvas, paint, m_posX, m_posY);
37          }
38      }
39      /**更新子弹的坐标点**/
40      public void UpdateBullet() {
41          if (mFacus) {
42              m_posY -= BULLET_STEP_Y;
43          }
44      }
45  }
```

新建敌机类 Enemy，与子弹类不同的地方在于，敌机类需要增加敌机死亡的判断及死亡动画的绘制。以下代码 30 行的变量 mState 用于记录当前敌机的状态，取值范围为 ENEMY_ALIVE_STATE 和 ENEMY_DEATH_STATE。同理，初始化的时候需要实例化两个动画，一个活着的动画，一个死亡动画，在播放的时候需要判断当前敌机的状态绘制相应的动画。在更新敌机的状态时，必须为当前敌机绘制完死亡动画后才能停止播放动画，如代码 63 行、64 行所示。

```
01  package guo.supermario.shooting;              //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容。
//……
08  public class Enemy {
09      /**敌机存活状态**/
10     public static final int ENEMY_ALIVE_STATE = 0;
11      /**敌机死亡状态**/
12     public static final int ENEMY_DEATH_STATE = 1;
13      /**敌机行走的 Y 轴速度**/
14      static final int ENEMY_STEP_Y = 5;
15      /**子弹图片的宽度**/
16      static final int BULLET_WIDTH = 40;
```

```
17      /** 子弹的 X,Y 坐标 **/
18      public int m_posX = 0;
19      public int m posY = 0;
20      /** 敌机行走的动画 **/
21      private Animation mAnimation0 = null;
22      /** 敌机死亡的动画 **/
23      private Animation mAnimation1 = null;
24      /**播放动画状态**/
25      public int mAnimState = 0;
26
27      /**是否更新绘制敌机**/
28      boolean mFacus = false;
29      /**敌机状态**/
30      int mState =0;
31      Context mContext = null;
32      public Enemy(Context context, Bitmap[] frameBitmap,Bitmap[]
        deadBitmap) {
33      mContext = context;
34      mAnimation0 = new Animation(mContext, frameBitmap, true);
35      mAnimation1 = new Animation(mContext, deadBitmap, false);
36      }
37      /**初始化坐标**/
38      public void init(int x, int y) {
39      m posX = x;
40      m_posY = y;
41      mFacus = true;
42      mAnimState = ENEMY ALIVE STATE;
43      mState = ENEMY ALIVE STATE;
44      mAnimation0.reset();
45      mAnimation1.reset();
46      }
47      /**绘制敌机动画**/
48      public void DrawEnemy(Canvas Canvas, Paint paint) {
49      if (mFacus) {
50          if(mAnimState == ENEMY ALIVE STATE) {
51           mAnimation0.DrawAnimation(Canvas, paint, m_posX, m_posY);
52          }else if(mAnimState == ENEMY DEATH STATE) {
53          mAnimation1.DrawAnimation(Canvas, paint, m posX, m posY);
54          }
55
56      }
57      }
58      /**更新敌机状态**/
59      public void UpdateEnemy() {
60      if (mFacus) {
61          m posY += ENEMY STEP Y;
62          //当敌机状态为死亡并且死亡动画播放完毕，不再绘制敌机
63          if(mAnimState == ENEMY_DEATH_STATE) {
64              if(mAnimation1.mIsend) {
65                  mFacus = false;
66                  mState = ENEMY_DEATH_STATE;
67              }
68          }
69      }
70      }
```

21.3　场景的绘制

（1）运行游戏的时候，首先呈现给用户的并不是游戏本身，而是一些供选择的菜单，用户可以通过菜单选项设置游戏的一些参数。然而在这里，我们只是简单地放置了一个开始游戏的按钮，如下所示，用户单击该按钮将跳转到游戏所在的界面 SurfaceViewActivity。

```
01  public class startActivity extends Activity {
02      Context mContext = null;
03      @Override
04      public void onCreate(Bundle savedInstanceState) {
05          super.onCreate(savedInstanceState);
06          setContentView(R.layout.main);
07          mContext = this;
08          /**开始**/
09          Button botton0 = (Button)findViewById(R.id.button0);
10          botton0.setOnClickListener(new OnClickListener() {
11              @Override
12              public void onClick(View arg0) {
13               Intent intent = new
                 Intent(mContext,SurfaceViewAcitvity.class);
14               startActivity(intent);
15              }
16          });
17
18      }
```

（2）新建 SurfaceViewActivity，如下所示，在 onCreate 中设置全屏，并获得 Display 变量，通过 Display 变量得到高和宽，然后实例化 AnimView 类并设置为当前显示的内容。

```
01  public class SurfaceViewAcitvity extends Activity{
02      AnimView mAnimView = null;
03      @Override
04      public void onCreate(Bundle savedInstanceState) {
05      super.onCreate(savedInstanceState);
06      //全屏显示窗口
07      requestWindowFeature(Window.FEATURE_NO_TITLE);
08      getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
09          WindowManager.LayoutParams.FLAG_FULLSCREEN);
10      //获取屏幕宽高
11      Display display = getWindowManager().getDefaultDisplay();
12
13      //显示自定义的游戏 View
14      mAnimView = new AnimView(this,display.getWidth(),
        display.getHeight());
15      setContentView(mAnimView);
16      }
```

（3）在 SurfaceViewActivity 中新建 AnimView 类继承于 SurfaceView，并实现 Callback 和 Runnale 接口，在类的开始定义了一些成员变量包括子弹类、敌机类、背景地图以及子弹的数量、敌机的数量、背景地图的初始坐标等等。在构造函数中，获得 SurfaceHolder，然后执行最关键的函数 init()，开始游戏。

```
001 public class AnimView extends SurfaceView implements Callback,Runnable{
```

```
002         /**屏幕的宽高**/
003 private int mScreenWidth = 0;
004 private int mScreenHeight = 0;
005 private int BgHeight = 0;
006 /**游戏主菜单状态**/
007 private  static final int STATE GAME = 0;
008
009 /**游戏状态**/
010 private int mState = STATE_GAME;
011
012 Paint mPaint = null;
013
014 /**游戏背景资源，两张图片进行切换让屏幕滚动起来**/
015 private Bitmap mBitMenuBG = null;
016
017 /**记录两张背景图片时时更新的 Y 坐标**/
018 private int mBitposY0 =0;
019 private int mBitposY1 =0;
020
021 /**飞机动画帧数**/
022 final static int PLAN ANIM COUNT = 6;
023
024 /**子弹动画帧数**/
025 final static int BULLET_ANIM_COUNT = 4;
026
027 /**子弹对象的数量**/
028 final static int BULLET POOL COUNT = 15 ;
029
030 /**飞机移动步长**/
031 final static int PLAN_STEP = 10;
032
033 /**每过 500 毫秒发射一颗子弹**/
034 final static int PLAN TIME = 500;
035
036 /**子弹图片向上偏移量**/
037 final static int BULLET_UP_OFFSET = 40;
038 /**子弹图片向左偏移量**/
039 final static int BULLET LEFT OFFSET = 5;
040
041 /**敌机对象的数量**/
042 final static int ENEMY_POOL_COUNT = 5 ;
043
044 /**敌机行走动画帧数**/
045 final static int ENEMY ALIVE COUNT = 1 ;
046 /**敌机死亡动画帧数**/
047 final static int ENEMY_DEATH_COUNT = 6 ;
048
049 /**敌机偏移量**/
050 final static int ENEMY POS OFF = 65 ;
051
052 /**游戏主线程**/
053 private Thread mThread = null;
054 /**线程循环标志**/
055 private boolean mIsRunning = false;
056
057 private SurfaceHolder mSurfaceHolder = null;
058 private Canvas mCanvas = null;
```

```
059
060 private Context mContext = null;
061
062
063 /**飞机动画**/
064 public Animation mAircraft =null;
065 /**飞机在屏幕中的坐标**/
066 public int mAirPosX = 0;
067 public int mAirPosY = 0;
068
069 /**敌机类**/
070 Enemy mEnemy[] = null;
071
072 /**子弹类**/
073 Bullet mBuilet[] = null;
074 Bitmap mBitbullet[] = null;
075
076 /**初始化发射子弹 ID 升序**/
077 public int mSendId = 0;
078
079 /**上一颗子弹发射的时间**/
080 public Long mSendTime = 0L;
081 /**手指在屏幕触摸的坐标**/
082 public int mTouchPosX = 0;
083 public int mTouchPosY = 0;
084
085 /**标志手指在屏幕触摸中**/
086 public boolean mTouching = false;
087
088 /**
089  * 构造方法
090  *
091  * @param context
092  */
093 public AnimView(Context context,int screenWidth, int screenHeight) {
094     super(context);
095     mContext = context;
096     mPaint = new Paint();
097     mScreenWidth = screenWidth;
098     mScreenHeight = screenHeight;
099     /**获取 mSurfaceHolder**/
100     mSurfaceHolder = getHolder();
101     mSurfaceHolder.addCallback(this);
102     setFocusable(true);
103     init();
104     setGameState(STATE_GAME);
105 }
```

（4）在初始化函数中，首先读取 R.drawable.map 作为背景图片，通过 getHeight()获得图片的高度，并初始化图片的坐标。这里我们采用两张背景地图拼接的做法，当一张往下移动的时候，上面会空出一部分，这部分就用同样的一张图片的下面去补全，这样看起来地图就是连贯变化的。这两张图片的坐标相差整张图片的高度值，初始化的时候一张放在屏幕的左上角，第二张放在第一张上面。

接着分别初始化敌机类、子弹类，将上一颗子弹发射时间初始化为当前时间。

```
01  private void init() {
```

```
02      /**游戏背景**/
03      mBitMenuBG = ReadBitMap(mContext,R.drawable.map);
04
05      /**创建主角飞机动画对象**/
06      mAircraft = new Animation(mContext,new int[]
        {R.drawable.plan_0,R.drawable.plan_1,R.drawable.
         plan_2,R.drawable.plan_3,R.drawable.plan_4,R.drawable.
         plan_5},true);
07
08      /**第一张图片在屏幕 0 点，第二张图片在第一张图片上方**/
09      mBitposY0 = 0;
10      mBitposY1 = - mBitMenuBG.getHeight();
11      BgHeight = mBitMenuBG.getHeight();
12      Log.e("guojs","ScreenHeight"+mScreenHeight);
13
14      /**初始化飞机的坐标**/
15      mAirPosX = 150;
16      mAirPosY = 400;
17
18      /**这里敌机行走动画就 1 帧**/
19      Bitmap []bitmap0 = new Bitmap[ENEMY_ALIVE_COUNT];
20      bitmap0[0] = ReadBitMap(mContext,R.drawable.enemy);
21      /**敌机死亡动画**/
22      Bitmap []bitmap1 = new Bitmap[ENEMY_DEATH_COUNT];
23      for(int i =0; i< ENEMY_DEATH_COUNT; i++) {
24          bitmap1[i] = ReadBitMap(mContext,R.drawable.bomb_enemy_0 + i);
25      }
26
27      /**创建敌机对象**/
28      mEnemy = new Enemy[ENEMY_POOL_COUNT];
29      for(int i =0; i< ENEMY_POOL_COUNT; i++) {
30          mEnemy[i] = new Enemy(mContext,bitmap0,bitmap1);
31          mEnemy[i].init(i * ENEMY_POS_OFF, 0);
32      }
33
34      /**创建子弹类对象**/
35      mBuilet = new Bullet[BULLET_POOL_COUNT];
36      mBitbullet = new Bitmap[BULLET_ANIM_COUNT];
37      for(int i=0; i<BULLET_ANIM_COUNT;i++) {
38          mBitbullet[i] = ReadBitMap(mContext,R.drawable.bullet);
39      }
40      for (int i =0; i< BULLET_POOL_COUNT;i++) {
41          mBuilet[i] = new Bullet(mContext,mBitbullet);
42      }
43      mSendTime = System.currentTimeMillis();
44  }
45  //设置游戏的状态
46  private void setGameState(int newState) {
47      mState =  newState;
48  }
49
50  /**
51   * 读取本地资源的图片
52   *
53   * @param context
54   * @param resId
55   * @return
56   */
```

```
57  public Bitmap ReadBitMap(Context context, int resId) {
58      BitmapFactory.Options opt = new BitmapFactory.Options();
59      opt.inPreferredConfig = Bitmap.Config.RGB 565;
60      opt.inPurgeable = true;
61      opt.inInputShareable = true;
62      // 获取资源图片
63      InputStream is = context.getResources().openRawResource(resId);
64      return BitmapFactory.decodeStream(is, null, opt);
65  }
```

（5）与 Activity 类似，SurfaceView 的生命周期中会调用一些函数，如下所示。在 SurfaceView 创建的时候，将执行 surfaceCreated()函数，在该函数中我们调用类中的线程，并执行。在 surfaceView 被销毁时，将线程执行标志位 mIsRunning 设置为 false，线程将停止执行。

```
01  //当 SurfaceView 的属性如高宽发生改变时触发
02  @Override
03  public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2,
04      int arg3) {
05      // surfaceView 的大小发生改变的时候
06
07  }
08  //当 surfaceView 被创建时触发
09  @Override
10  public void surfaceCreated(SurfaceHolder arg0) {
11      /**启动游戏主线程**/
12      mIsRunning = true;
13      mThread = new Thread(this);
14      mThread.start();
15  }
16  // surfaceView 销毁时触发
17  @Override
18  public void surfaceDestroyed(SurfaceHolder arg0) {
19      mIsRunning = false;
20  }
```

（6）线程执行时需要给变量 mSurfaceHolder 加上安全锁，防止其他函数去操作这个变量。接着锁定调用 lockCanvas()函数获得锁定的画布，绘制场景，最后调用 unlockCanvasAndPost 解锁并将画面显示到屏幕上。

绘制场景的函数实现如以下代码 019～026 行所示，而场景的绘制函数又分为两部分：renderBg()和 updateBg()。renderBg()函数主要是绘制动画，而 updateBg()函数则是更新场景的一些相关的参数，如背景图片的坐标 mBitposY0 和 mBitposY1、飞机的坐标 mAirPosY 和 mAirPosX。同时要处理敌机和子弹的初始化，首先执行敌机坐标和状态的更新，再判断坐标是否超出屏幕之外以及是否死亡，来决定是否重新初始化相应的敌机。子弹与敌机略有不同，子弹每隔一个固定时间则初始化一次，因为子弹对象的总数是确定的，因此这个初始化也是循环进行的。

除此之外，还要判断敌机与子弹的碰撞情况，调用函数 Collision()用来循环遍历所有敌机和子弹，将它们的碰撞抽象成点与线的交集，当子弹的左上角坐标位于敌机图片上方时表明敌机被击中，则更改敌机状态为死亡。当然，这个判定方式有个很明显的缺陷，就是算法效率太低，还有一点不严密，但是作为演示，只为了将这种使用方式告诉大家，大家可以自行完善这个程序。

```
001 @Override
002 public void run() {
003     while (mIsRunning) {
004     //在这里加上线程安全锁
005     synchronized (mSurfaceHolder) {
006         /**拿到当前画布，然后锁定**/
007         mCanvas =mSurfaceHolder.lockCanvas();
008         Draw();
009         /**绘制结束后解锁显示在屏幕上**/
010         mSurfaceHolder.unlockCanvasAndPost(mCanvas);
011     }
012     try {
013         Thread.sleep(100);
014     } catch (InterruptedException e) {
015         e.printStackTrace();
016     }
017     }
018 }
019 protected void Draw() {
020     switch (mState) {
021     case STATE_GAME:
022         renderBg();
023         updateBg();
024         break;
025     }
026 }
027 /** 绘制游戏地图 **/
028 public void renderBg() {
029 
030     mCanvas.drawBitmap(mBitMenuBG, 0, mBitposY0, mPaint);
031     mCanvas.drawBitmap(mBitMenuBG, 0, mBitposY1, mPaint);
032     /**绘制飞机动画**/
033     mAircraft.DrawAnimation(mCanvas, mPaint, mAirPosX, mAirPosY);
034 
035     /**绘制子弹动画*/
036     for (int i =0; i < BULLET_POOL_COUNT; i++) {
037        mBuilet[i].DrawBullet(mCanvas, mPaint);
038     }
039 
040     /**绘制敌机动画**/
041     for(int i =0; i< ENEMY_POOL_COUNT; i++) {
042         mEnemy[i].DrawEnemy(mCanvas, mPaint);
043     }
044 }
045 private void updateBg() {
046     /** 更新游戏场景的参数 **/
047     mBitposY0 += 10;
048     mBitposY1 += 10;
049     if (mBitposY0 == BgHeight) {
050         mBitposY0 = - BgHeight;
051     }
052     if (mBitposY1 == BgHeight) {
053         mBitposY1 = - BgHeight;
054     }
055 
056     /** 手指触摸屏幕更新飞机坐标 **/
057     if (mTouching) {
058 
```

```
059     if (mAirPosX < mTouchPosX) {
060         mAirPosX += PLAN STEP;
061     } else {
062         mAirPosX -= PLAN STEP;
063     }
064     if (mAirPosY < mTouchPosY) {
065         mAirPosY += PLAN STEP;
066     } else {
067         mAirPosY -= PLAN STEP;
068     }
069
070     if (Math.abs(mAirPosX - mTouchPosX) <= PLAN_STEP) {
071         mAirPosX = mTouchPosX;
072     }
073     if (Math.abs(mAirPosY - mTouchPosY) <= PLAN_STEP) {
074         mAirPosY = mTouchPosY;
075     }
076     }
077     /** 更新子弹动画 **/
078     for (int i = 0; i < BULLET POOL COUNT; i++) {
079     /** 子弹出屏后重新赋值**/
080         mBuilet[i].UpdateBullet();
081
082     }
083     /**绘制敌机动画**/
084     for(int i =0; i< ENEMY POOL COUNT; i++) {
085         mEnemy[i].UpdateEnemy();
086     /**敌机死亡或者敌机超过屏幕还未死亡，重置坐标**/
087     if(mEnemy[i].mState == Enemy.ENEMY_DEATH_STATE || mEnemy[i].m_posY
        >=mScreenHeight) {
088          mEnemy[i].init(UtilRandom(0,ENEMY_POOL_COUNT) *ENEMY_POS_OFF,
0);
089     }
090
091     }
092
093     /**根据时间初始化发射的子弹**/
094     if (mSendId < BULLET_POOL_COUNT) {
095     long now = System.currentTimeMillis();
096     if (now - mSendTime >= PLAN_TIME) {
097         mBuilet[mSendId].init(mAirPosX - BULLET_LEFT_OFFSET, mAirPosY -
            BULLET_UP_OFFSET);
098         mSendTime = now;
099         mSendId++;
100     }
101     }else {
102     mSendId = 0;
103     }
104
105     //更新子弹与敌机的碰撞
106     Collision();
107
108 }
109 /**
110     * 返回一个随机数
111     * @param botton
112     * @param top
113     * @return
114     */
```

```
115 private int UtilRandom(int botton, int top) {
116     return ((Math.abs(new Random().nextInt()) % (top - botton)) +
        botton);
117     }
118 public void Collision() {
119     //更新子弹与敌机的碰撞
120     for (int i = 0; i < BULLET_POOL_COUNT; i++) {
121     for (int j = 0; j < ENEMY_POOL_COUNT; j++) {
122        if(mBuilet[i].m_posX >= mEnemy[j].m_posX && (mBuilet[i].m_posX <=
           mEnemy[j].m_posX + 20)
123               &&mBuilet[i].m_posY >= mEnemy[j].m_posY &&
                  (mBuilet[i].m_posY<=mEnemy[j].m_posY + 20)
124
125        ) {
126            mEnemy[j].mAnimState = Enemy.ENEMY_DEATH_STATE;
127        }
128     }
129
130     }
131 }
```

（7）在类 SurfaceViewActivity 中还要增加 onTouchEvent()回调函数，用于处理用户单击触摸屏的事件，如以下代码 11～29 行所示。当触摸按下和放开时将调用类 AnimView 中的 UpdateTouchEvent()函数更新飞机的坐标。

```
01  public void UpdateTouchEvent(int x, int y, boolean touching) {
02      //在这里检测按钮按下播放不同的特效
03      switch (mState) {
04      case STATE_GAME:
05      mTouching = touching;
06      mTouchPosX = x;
07      mTouchPosY = y;
08      break;
09      }
10  }
11   public boolean onTouchEvent(MotionEvent event) {
12      //获得触摸的坐标
13      int x = (int) event.getX();
14      int y = (int) event.getY();
15      switch (event.getAction()) {
16      //触摸屏幕时刻
17      case MotionEvent.ACTION_DOWN:
18         mAnimView.UpdateTouchEvent(x, y,true);
19         break;
20      //触摸并移动时刻
21      case MotionEvent.ACTION_MOVE:
22         break;
23      //终止触摸时刻
24      case MotionEvent.ACTION_UP:
25         mAnimView.UpdateTouchEvent(x, y,false);
26         break;
27      }
28      return false;
29    }
```

21.4 知识拓展

我们在绘制游戏场景的时候用到了 synchronized (mSurfaceHolder)，那么这里的 synchronized 作用是什么呢？这的 mSurfaceHolder 是监视器要监视的对象，当 mSurfaceHolder 被监视器监视的时候，同一时刻只能有一个线程访问它，其他要访问它的线程必须在等待队列中等待。

下面我们来看一个小例子，如下所示，这里的 execute()方法前面没有用 synchronized 修饰，因此输出处的数字结果交错在一起，说明不是同步的，两个方法在不同的线程中是异步调用的。

```
01  package com.supermario.activitytest;                    //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容
//……
08  public class ActivityTest extends Activity {
09      /** Called when the activity is first created. */
10      private static String TAG="ActivityTest";
11      @Override
12      //程序创建
13      public void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.main);
16          TestThread test=new TestThread();
17          Runnable runabble=new TestThread2(test);
18          Thread a=new Thread(runabble,"A");
19          a.start();
20          Thread b=new Thread(runabble,"B");
21          b.start();
22      }
23      public class TestThread{
24          public void execute(){            //synchronized，未修饰
25          for(int i=0;i<1000;i++){
26              Log.e(TAG,i+"");
27          }
28          }
29      }
30      class TestThread2 implements Runnable{
31          TestThread test=null;
32
33          public TestThread2(TestThread pTest){    //对象有外部引入，这样保证
            是同一个对象
34              test=pTest;
35          }
36
37           public void run(){
38              test.execute();
39           }
40      }
41  }
```

在上面的代码中如果在代码 24 行的函数前面添加 synchronized 修饰，输出的数字是有序的，则首先输出 A 的数字，然后是 B，说明线程是同步的，虽然是不同的线程，但两个

方法是同步调用的。上面虽然是两个不同的线程，但是是同一个实例对象。下面使用不同的实例对象进行测试。

如下所示，输出的数字将交错在一起，说明虽然增加了 synchronized 关键字来修饰方法，但是不同的线程调用各自的对象实例，两个方法仍然是异步的。

```
01  package com.supermario.activitytest;                //声明包语句
02~07 行为引入相关类，这里不再列举，请阅读光盘内容
//……
08  public class ActivityTest extends Activity {
09      /** Called when the activity is first created. */
10      private static String TAG="ActivityTest";
11      @Override
12      //程序创建
13      public void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.main);
16          Runnable runabble=new TestThread2();
17          Thread a=new Thread(runabble,"A");
18          a.start();
19          Thread b=new Thread(runabble,"B");
20          b.start();
21      }
22      public class TestThread{
23          public synchronized void  execute(){         //synchronized 同
            步的
24          for(int i=0;i<1000;i++){
25              Log.e(TAG,i+"");
26          }
27          }
28      }
29      class TestThread2 implements Runnable{
30           public void run(){
31               TestThread test = new TestThread();       //每次创建一个新的
                 实例对象
32               test.execute();
33           }
34      }
```

对于这种多个实例，要想实现同步即输出的数字是有序并且按线程的先后顺序输出，我们可以增加一个静态变量，对它进行加锁，就像本章中使用的方式一样。将上面代码 22～28 行改为下面的形式。这样就保证了仅有一个 lock 对象的实例，谁握有这个锁就可以执行其中的操作。需要注意的是，这里的对象必须是静态的，否则不同的实例线程仍是不安全的。

```
01  public class TestThread{
02      private static Object lock=new Object();            //必须是静态的
03      public synchronized void  execute(){
04        synchronized(lock){
05            for(int i=0;i<1000;i++){
06                Log.e(TAG,i+"");
07            }
08        }
09      }
10  }
```

21.5　本 章 小 结

本章主要介绍了开发一款射击游戏的基本思路，包括如何让飞机“飞起来”，如何处理子弹和敌机的碰撞，如何更新整个游戏场景的元素等。大家除了要明白射击游戏的基本原理，更重要的是熟悉 SurfaceView 的使用，理解动画的原理。

第 22 章　3D 迷宫游戏

Android 手机一般都带有加速度传感器，利用这个传感器我们可以开发出相应的游戏，而且通过该传感器控制游戏的进行，可以获得与平常游戏不一样的体验。本章将结合加速度传感器开发一款 3D 迷宫游戏。

22.1　游戏地图绘制方法

记得小时候玩游戏，就一直很好奇游戏中千变万化、形式各异的地图是如何产生的。其实地图的绘制原理很简单，首先游戏地图分为可通过元素和不可通过元素，通过这些元素组成了地图的基本骨架，接着在这骨架之上铺上“瓷砖”，就产生了丰富多彩的画面。绘制地图可以通过软件来完成，接下来我们将简单介绍 Mappy 地图编辑器的使用。

如图 22.1 所示，打开 Mappy 地图编辑器并新建地图。地图由一个个“瓷砖”铺砌而成，我们一般设置“瓷砖”大小为 32×32，然后根据模拟器分辨率的大小计算出地图的宽、高各需要多少块“瓷砖”。

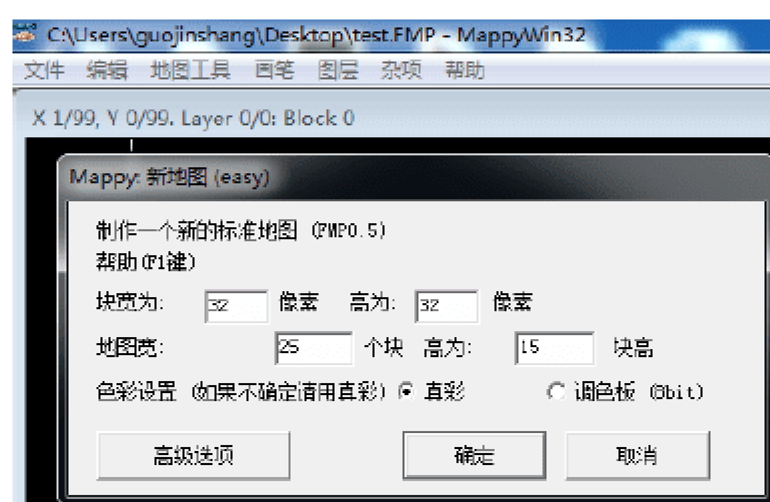


图 22.1　新建地图

选择文件菜单下面的“导入图块”命令，选择一张 PNG 格式的图片，如图 22.2 所示，可以看到我们导入的图片被切割成一个个小“瓷砖”，每个“瓷砖”的大小就是 32×32。接着，我们可以选中右侧的图块，并在左边将图块“贴”到地图上。

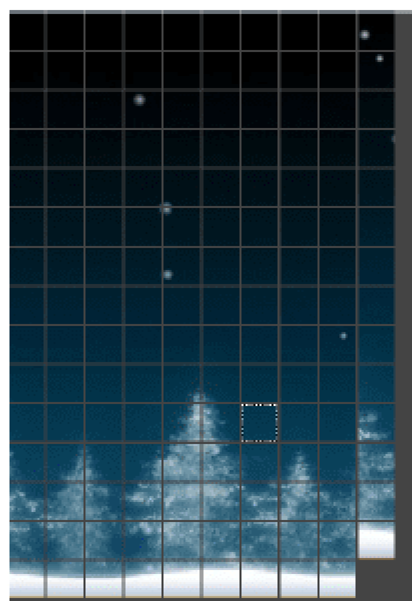


图 22.2　导入图块

每一个地图一般由 3 个图层组成：最底层、实体层和物理层。最底层绘制游戏的背景，一般绘制的时候首先绘制该层，接着在最底层的基础上绘制实体层如墙壁、树木等，最后绘制物理层。物理层与实体层绘图位置一致，只是不需要图片填充。完成了这 3 层的绘制之后将地图保存成 FMP 格式，并导出 txt 数据，txt 中通过数组将地图的 3 层表示出来。

地图的数据已经有了，接下来就是要在代码中将地图的原型展现出来。首先在代码中将图片资源载入，接着根据地图中的数据获得“瓷砖”在图片资源中的 X、Y 坐标，再依次将获得的一张张小图片贴到指定的位置中，完成地图数据的复原。在地图中通过第三层数据可以判断障碍物的位置，也就可以确认哪些位置可以通过，哪些位置不可以通过。

以上简单介绍了地图编辑器的原理和使用方法，上面绘制地图的方式可以用于绝大多数地图的绘制。

22.2　游戏地图的绘制

本章游戏需要用到一张张的迷宫地图来作为关卡，如图 22.3 所示，地图中需要绘制的元素包括小球、墙壁、地板等。下面我们依次介绍每个元素的绘制方法。

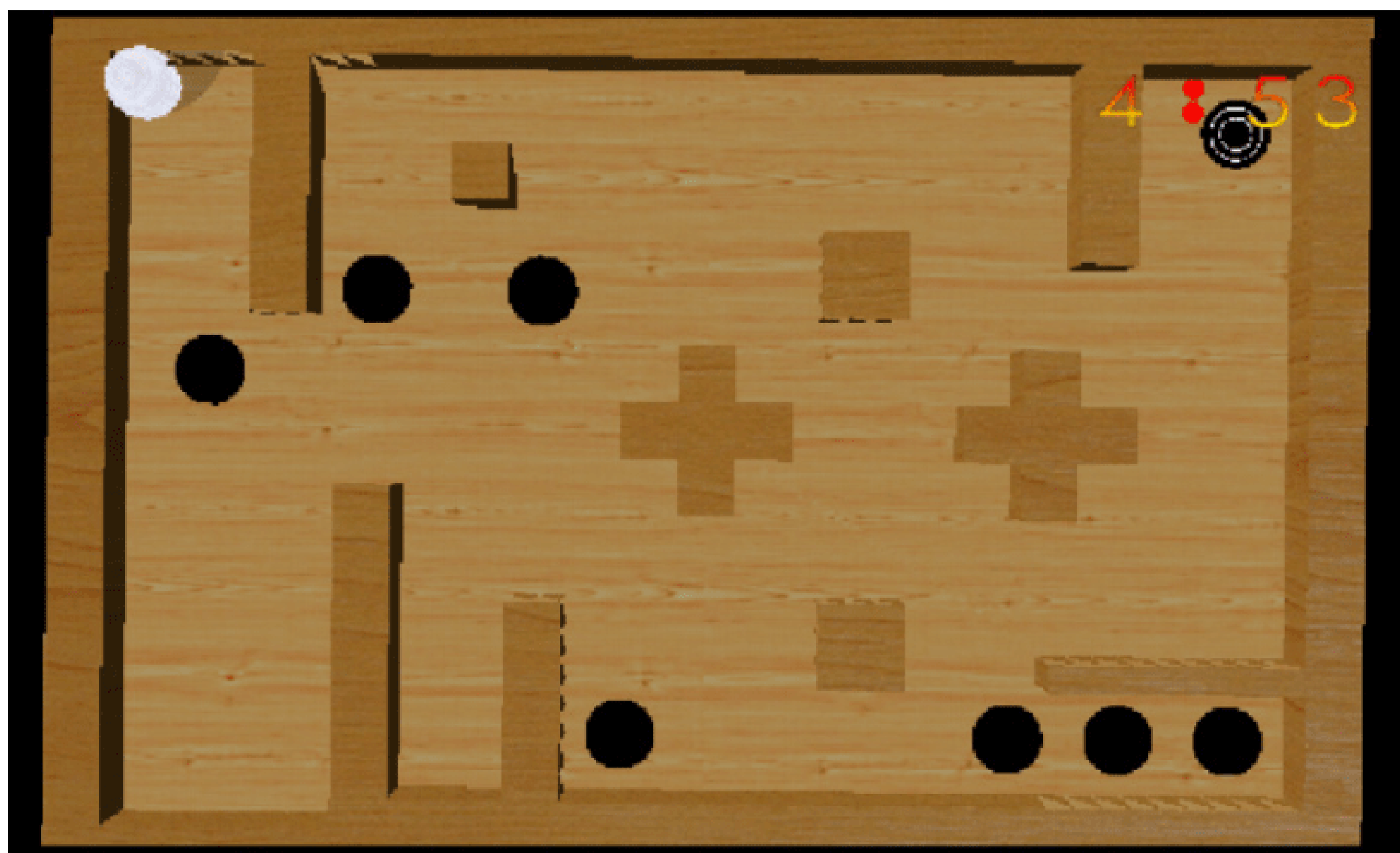


图 22.3　游戏关卡地图

22.2.1　3D 绘图基本知识

可能还有一些读者之前并未接触过 3D 绘图，为了方便以下的讲解，下面先来普及一下 3D 绘图的一些基本知识。3D 图像的最小单位称为点（point）或者顶点 vertex，它们代表三维空间中的一个点并用来建造更复杂的物体。多边形就是由点构成的，而物体由多个多边形组成。尽管通常 OpenGL 支持多种多边形，但 OpenGLEs 只支持三边形（即三角形），所以即使我们要绘制一个正方形也要把它拆分为两个三角形来绘制。

默认情况下，以屏幕中心为坐标轴原点。原点左方 x 为负值，右边为正值。原点上方 y 为正值，原点下方为负值。垂直屏幕向外 z 为正值，垂直屏幕向里 z 为负值。默认情况

下，从原点到屏幕边缘为 1.0f，沿各轴增加或减小的数值是以任意刻度进行的，它们不代表任何真实单位，如英尺、像素或米等。你可以选择任何对你的程序有意义的刻度（全局必须保持单位一致，不能一部分使用米，一部分使用像素）。OpenGL 只是将它作为一个参照单位处理，保证它们具有相同的距离，如图 22.4 所示。

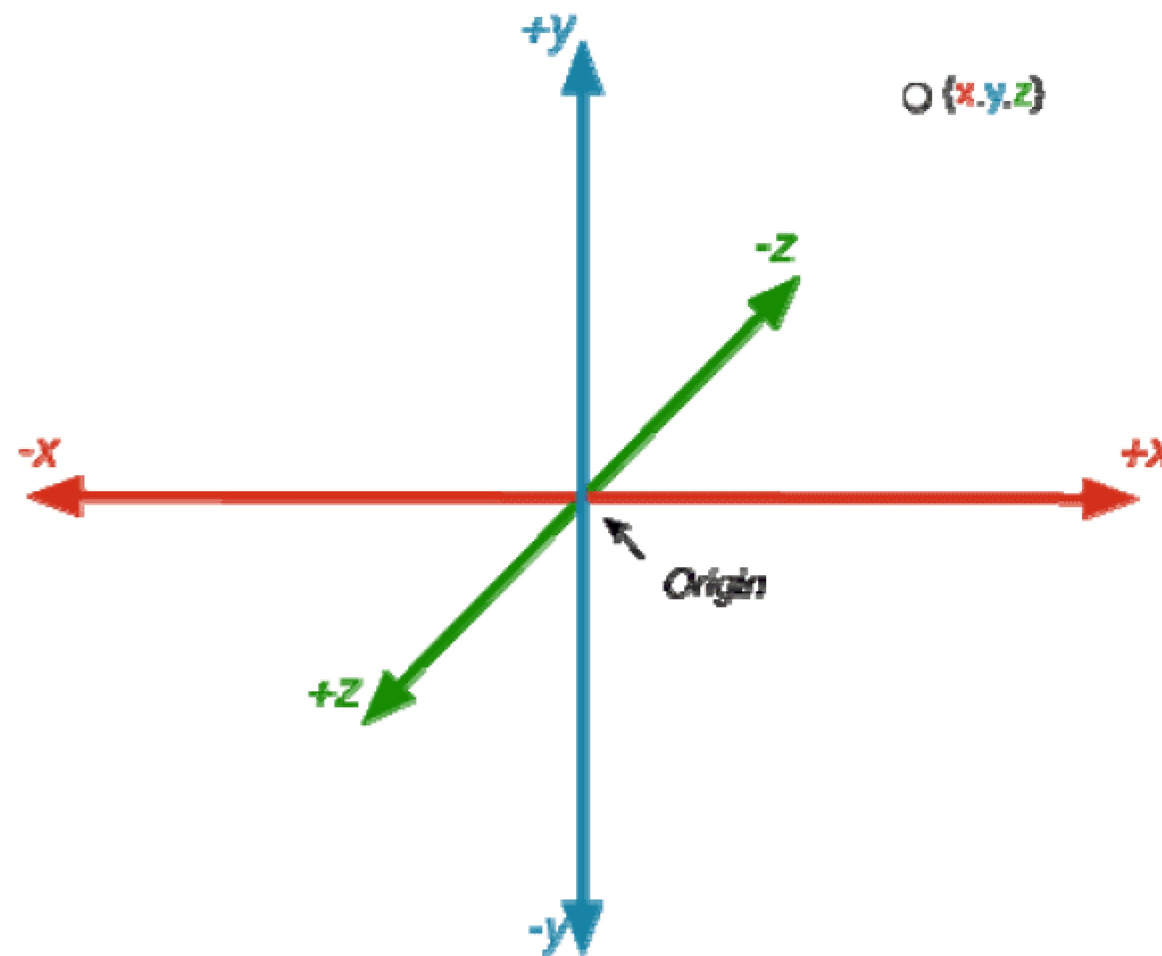


图 22.4　3D 坐标系

了解了坐标轴，我们来看看如何在坐标系中表示一个点。通常用一组浮点数来表示点，例如一个正方形的 4 个顶点可表示为：

```
1   float vertices[] = {
2       -1.0f, 1.0f, 0.0f,        //左上
3       -1.0f, -1.0f, 0.0f,       //左下
4       1.0f, -1.0f, 0.0f,        //右下
5       1.0f, 1.0f, 0.0f,         //右上
6   };
```

为了提高性能，通常还需要将浮点数组存入一个字节缓冲中，所以有了下面的操作：

```
1   ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
                                     //申请内存
2   vbb.order(ByteOrder.nativeOrder());  //设置字节顺序，其中
    ByteOrder.nativeOrder()是获取本机字节顺序
3   FloatBuffer vertexBuffer = vbb.asFloatBuffer();
                                   //转换为 float 型
4   vertexBuffer.put(vertices);        //添加数据
5   vertexBuffer.position(0);          //设置缓冲区起始位置
```

OpenGLES 的很多函数功能的使用状态是处于关闭的，启用和关闭这些函数可以用 glEnableClientState、glDisableClientState 来完成。

```
1   //指定需要启用定点数组
2   gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
3   //说明启用数组的类型和字节缓冲，类型为 GL_FLOAT
4   gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
5   //不再需要时，关闭顶点数组
6   gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
```

22.2.2　地板

如下所示为地板类的实现代码，在构造函数中根据传入的高宽数值初始化地板的顶点坐标缓冲、纹理坐标缓冲和法向量缓冲。其中地板块划分成两个三角形，也就是有 6 个顶点，相应的纹理采用平铺的方式铺在上面，而地板的正方向均设置为（0，1，0）。

通过以上的设置，最终达到的效果是将一个表示地板纹理的图片平铺到指定的顶点围成的面上，也就是游戏界面上。

```
001 //表示地板的类
002 public class Floor {
003     //顶点坐标数据缓冲
004     private FloatBuffer   mVertexBuffer;
005     //顶点纹理数据缓冲
006     private FloatBuffer mTextureBuffer;
007     //顶点法向量缓冲类
008     private FloatBuffer mNormalBuffer;
009     //顶点数量
010     int vCount=0;
011     //地板横向 width 个单位
012     int width;
013     //地板纵向 height 个单位
014     int height;
015     //构造函数
016     public Floor(int width,int height)
017     {
018         this.width=width;
019         this.height=height;
020         //顶点坐标数据的初始化——begin
021         //每个地板块 6 个顶点
022         vCount=6;
023         float []vertices=new float[]
024         {
025                 -width*UNIT_SIZE/2,0,-height*UNIT_SIZE/2,
026                 -width*UNIT_SIZE/2,0,height*UNIT_SIZE/2,
027                 width*UNIT_SIZE/2,0,height*UNIT_SIZE/2,
028
029                 width*UNIT_SIZE/2,0,height*UNIT_SIZE/2,
030                 width*UNIT_SIZE/2,0,-height*UNIT_SIZE/2,
031                 -width*UNIT_SIZE/2,0,-height*UNIT_SIZE/2
032         };
033         //分配空间
034         ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
035         //设置字节顺序
036         vbb.order(ByteOrder.nativeOrder());
037         //转换为 float 型缓冲
038         mVertexBuffer = vbb.asFloatBuffer();
039         //向缓冲区中放入顶点坐标数据
040         mVertexBuffer.put(vertices);
041         //设置缓冲区起始位置
042         mVertexBuffer.position(0);
043         float textures[]=new float[]
044         {
045                 0,0,
```

```
046                 0,2,
047                 2,2,
048
049                 2,2,
050                 2,0,
051                 0,0
052           };
053           //顶点数据的初始化——end
054           //顶点纹理数据的初始化——begin
055           //分配空间
056           ByteBuffer tbb = ByteBuffer.allocateDirect(textures.length*4);
057           //设置字节顺序
058           tbb.order(ByteOrder.nativeOrder());
059           //转换为 Float 型缓冲
060           mTextureBuffer= tbb.asFloatBuffer();
061           //向缓冲区中放入顶点纹理数据
062           mTextureBuffer.put(textures);
063           //设置缓冲区起始位置
064           mTextureBuffer.position(0);
065           //由于不同平台字节顺序不同，数据单元不是字节的一定要经过 ByteBuffer
066           //转换，关键是要通过 ByteOrder 设置 nativeOrder()，否则有可能会出问题
067           //顶点纹理数据的初始化——end
068
069           //顶点的初始化——begin
070           float normals[]=new float[vCount*3];
071           for(int i=0;i<vCount;i++)
072           {
073               normals[i*3]=0;
074               normals[i*3+1]=1;
075               normals[i*3+2]=0;
076           }
077           //分配空间
078           ByteBuffer nbb = ByteBuffer.allocateDirect(normals.length*4);
079           //设置字节顺序
080           nbb.order(ByteOrder.nativeOrder());
081           //转换为 float 型缓冲
082           mNormalBuffer = nbb.asFloatBuffer();
083           //向缓冲区中放入顶点法向量数据
084           mNormalBuffer.put(normals);
085           //设置缓冲区起始位置
086           mNormalBuffer.position(0);
087           //顶点法向量数据的初始化——end
088       }
089       //绘制地板
090       public void drawSelf(GL10 gl,int texId)
091       {
092           //为画笔指定顶点坐标数据
093           gl.glVertexPointer
094           (
095                   3,                  //每个顶点的坐标数量为 3，xyz
096                   GL10.GL_FLOAT,      //顶点坐标值的类型为 GL_FLOAT
097                   0,                  //连续顶点坐标数据之间的间隔
098                   mVertexBuffer       //顶点坐标数据
099           );
100
101           //为画笔指定纹理 ST 坐标缓冲
102           gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTextureBuffer);
```

```
103         //绑定当前纹理
104         gl.glBindTexture(GL10.GL_TEXTURE_2D, texId);
105         //设置法向量
106         gl.glNormalPointer(GL10.GL_FLOAT, 0, mNormalBuffer);
107         //绘制图形
108         gl.glDrawArrays
109         (
110                 GL10.GL_TRIANGLES,     //以三角形方式填充
111                 0,                     //开始点编号
112                 vCount                 //顶点的数量
113         );
114     }
115 }
```

22.2.3　墙壁

和地板类类似，墙壁类 wall 的基本思路也是在构造函数中将顶点坐标数据、顶点法向量数据、顶点纹理数据存放到相应的缓冲变量中，接着使用 drawself 将图形绘出。然而，墙壁类比地板类要复杂，地板直接平铺即可，而墙壁需要判断哪些位置需要放墙壁，哪些位置不用。

（1）如下所示，在类的开始先声明 3 个缓冲变量，同时声明二维数组 indexFlag 用于存放当前地图每一点的扫描情况。

```
01  //顶点坐标数据缓冲
02  private FloatBuffer   mVertexBuffer;
03  //顶点纹理数据缓冲
04  private FloatBuffer   mTextureBuffer;
05  //顶点法向量数据缓冲
06  private FloatBuffer   mNormalBuffer;
07  //顶点数量
08  int vCount;
09  //用于记录当前点是否扫描过，“1”表示此点不需要再扫描，“0”表示此点需要扫描
10  private int[][] indexFlag;
```

（2）接着我们需要根据地图的数据生成相应的墙壁，如下所示，地图的数据是由 0 和 1 组成的数组表示，1 代表墙壁，0 代表地板，因此我们需要利用算法将 1 的部分转化成墙壁。

```
01  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
02  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
03  {1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
04  {1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
05  {1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
06  {1,1,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
07  {1,1,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
08  {1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
09  {1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
10  {1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
```

（3）如下所示，函数 returnMaxBlock()的用途在于返回当前点周围最大的面积数，传入的点必须为墙壁所在的点。在函数中将扫描传入点所在行，并获得最大连续的列数，传入 area[1][0]。由于我们是逐行扫描，因此连续的最大行数为 1。最后，为了提高扫描的效

率，那些被扫描过的点将被做上标记 1，下次不再被扫描。

```
01  //根据当前点，判断出此点周围最大的面积块数，当前点必须为墙
02    public int[][] returnMaxBlock(int rowIndex,int colIndex)
03    {
04      int rowindex=rowIndex;
05      int colindex=colIndex;
06      int rowsize;//用于记录行的大小
07      int colsize;//用于记录列的大小
08
09      int [][] area=new int[2][2];//用于记录位置，area[0]表示起始点索引，
        area[1]表示长度和宽度
10      area[0][0]=rowIndex;area[0][1]=colIndex;
11      //横向长度为 1
12      int tempRowSize=1;  //宽度
13      int tempColSize=1;  //长度
14      //从该索引点往右遍历，直到遇到 0，得到长度
15      while(colindex+1<MAP[0].length&&MAP[rowindex][colindex+1]
        ==1&&indexFlag[rowindex][colindex+1]==0)
16      {
17          tempColSize++;   //长度加一
18          colindex++;      //列索引加一
19      }
20      //存放最后索引点的位置
21      rowsize=tempRowSize;colsize=tempColSize;
22      area[1][0]=colsize;area[1][1]=rowsize;
23      //将 indexFlag 扫描过的格子置为 1
24      for(int i=area[0][0];i<area[0][0]+area[1][1];i++)
25      {
26          for(int j=area[0][1];j<area[0][1]+area[1][0];j++)
27          {
28              indexFlag[i][j]=1;//将其值设置为 1，表示不用再扫描
29          }
30      }
31      return area;
32    }
```

（4）最关键的地方在于该类的构造函数，需要计算出墙壁所有顶点的坐标、纹理坐标、法向量坐标，为接下去进行绘制提供数据。从以下代码 015 行开始对地图的数组进行扫描，每当扫描到一个点值为 1 时，就调用函数 returnMaxBlock()获得当前点所在行中的连续块坐标信息 area，接着对 area 中的每一个元素，即每一个“方块”计算上、前、后、左、右共 5 个面的所有顶点、纹理和法向量数据，并保存到数组中。

```
001   //构造函数
002   public Wall()
003   {
004    //顶点坐标数据的初始化——begin
005      int rows=MAP.length;
006      int cols=MAP[0].length;
007      indexFlag=new int[rows][cols];
008      //顶点数组，用于存放顶点
009      ArrayList<Float> alVertex=new ArrayList<Float>();
010      //法向量数组，用于存放法向量
011      ArrayList<Float> alNormal=new ArrayList<Float>();
012      //纹理数组，用于存放纹理数据
013      ArrayList<Float> alTexture=new ArrayList<Float>();
```

```
014         //行扫描
015         for(int i=0;i<rows;i++)
016         {
017          //列扫描
018          for(int j=0;j<cols;j++)
019          {//对地图中的每一块进行处理
020              if(MAP[i][j]==1)               //当前点为墙
021              {
022                  // area[0]表示起始点行、列，area[1]表示宽度和高度
023                  int [][] area=returnMaxBlock(i,j);
024                  //对区域内的每个点建造围墙
025                  for(int k=area[0][0];k<area[0][0]+area[1][1];k++)
026                  {
027                      for(int t=area[0][1];t<area[0][1]+area[1][0];t++)
028                      {
029                          //建造顶层墙
030                          float xx1=t*UNIT_SIZE;          //1
031                          float y=FLOOR_Y+WALL_HEIGHT;
032                          float zz1=k*UNIT_SIZE;
033
034                          float xx2=t*UNIT_SIZE;          //2
035                          float zz2=(k+1)*UNIT_SIZE;
036
037                          float xx3=(t+1)* UNIT_SIZE;     //3
038                          float zz3=(k+1)*UNIT_SIZE;
039
040                          float xx4=(t+1)*UNIT_SIZE;      //4
041                          float zz4=k*UNIT_SIZE;
042                          //构造三角形
043                          alVertex.add(xx1);alVertex.add(y);alVertex.
                             add(zz1);
044                          alVertex.add(xx2);alVertex.add(y);alVertex.
                             add(zz2);
045                          alVertex.add(xx3);alVertex.add(y);alVertex.
                             add(zz3);
046
047                          alVertex.add(xx3);alVertex.add(y);alVertex.
                             add(zz3);
048                          alVertex.add(xx4);alVertex.add(y);alVertex.
                             add(zz4);
049                          alVertex.add(xx1);alVertex.add(y);alVertex.
                             add(zz1);
050
051                          //添加纹理，整块平铺
052                          alTexture.add((float)((float)t/cols));alTexture.
                             add((float)k/rows);
053                          alTexture.add((float)((float)t/cols));alTexture.
                             add((float)((float)(k+1)/rows));
054                          alTexture.add((float)((float)(t+1)/cols));
                             alTexture.add((float)((float)(k+1)/rows));
055
056                          alTexture.add((float)((float)(t+1)/cols));
                             alTexture.add((float)((float)(k+1)/rows));
057                          alTexture.add((float)((float)(t+1)/cols));
                             alTexture.add((float)k/rows);
058                          alTexture.add((float)((float)t/cols));
                             alTexture.add((float)k/rows);
```

```
059
060
061
062                 //建造向量
063                 alNormal.add(0f);alNormal.add(1f);
                    alNormal.add(0f);
064                 alNormal.add(0f);alNormal.add(1f);
                    alNormal.add(0f);
065                 alNormal.add(0f);alNormal.add(1f);
                    alNormal.add(0f);
066
067                 alNormal.add(0f);alNormal.add(1f);
                    alNormal.add(0f);
068                 alNormal.add(0f);alNormal.add(1f);
                    p1p1opvalNormal.add(0f);
069                 alNormal.add(0f);alNormal.add(1f);
                    alNormal.add(0f);
070
071                 //建造墙的上面
072                 if(k==0||MAP[k-1][t]==0)
073                 {
074                     float x1=t*UNIT_SIZE;
075                     float y1=FLOOR_Y;
076                     float z1=k*UNIT_SIZE;   //1
077
078                     float x2=t*UNIT_SIZE;
079                     float y2=FLOOR_Y+WALL_HEIGHT;
080                     float z2=k*UNIT_SIZE;    //2
081
082                     float x3=(t+1)*UNIT_SIZE;
083                     float y3=FLOOR_Y+WALL_HEIGHT;
084                     float z3=k*UNIT_SIZE;    //3
085
086                     float x4=(t+1)*UNIT_SIZE;
087                     float y4=FLOOR_Y;
088                     float z4=k*UNIT_SIZE;    //4
089                     //建造三角形
090                     alVertex.add(x1);alVertex.add(y1);alVertex.
                        add(z1);
091                     alVertex.add(x2);alVertex.add(y2);alVertex.
                        add(z2);
092                     alVertex.add(x3);alVertex.add(y3);alVertex.
                        add(z3);
093
094                     alVertex.add(x3);alVertex.add(y3);alVertex.
                        add(z3);
095                     alVertex.add(x4);alVertex.add(y4);alVertex.
                        add(z4);
096                     alVertex.add(x1);alVertex.add(y1);alVertex.
                        add(z1);
097                     //建造纹理
098                     alTexture.add((float)((float)(t-area[0][1])/
                        cols));alTexture.add(0f);
099                     alTexture.add((float)((float)(t-area[0][1])
                        /cols));alTexture.add((float)
                        ((float)1/rows));
100                     alTexture.add((float)((float)(t+1-area[0]
                        [1])/cols));alTexture.add((float))
                        ((float)1/rows));
101
```

```
102                         alTexture.add((float)((float)(t+1-area[0][1])
                           /cols));alTexture.add((float)((float)1
                           /rows));
103                        alTexture.add((float)((float)(t+1-area[0][1]
                           )/cols));alTexture.add(0f);
104                        alTexture.add((float)((float)(t-area[0][1])
                           /cols));alTexture.add((float)1/rows);
105                            //建造向量
106                        alNormal.add(0f);alNormal.add(0f);alNormal
                          .add(-1f);
107                        alNormal.add(0f);alNormal.add(0f);alNormal
                          .add(-1f);
108                        alNormal.add(0f);alNormal.add(0f);alNormal
                          .add(-1f);
109
110                        alNormal.add(0f);alNormal.add(0f);alNormal
                          .add(-1f);
111                        alNormal.add(0f);alNormal.add(0f);alNormal
                          .add(-1f);
112                        alNormal.add(0f);alNormal.add(0f);alNormal
                          .add(-1f);
113                    }
114                    //建造墙的下面
115                    if(k==rows-1||MAP[k+1][t]==0)
116                    {
117                        float x2=t*UNIT_SIZE;
118                        float y2=FLOOR_Y;
119                        float z2=(k+1)*UNIT_SIZE;
120
121                        float x1=t*UNIT_SIZE;
122                        float y1=FLOOR_Y+WALL_HEIGHT;
123                        float z1=(k+1)*UNIT_SIZE;
124
125                        float x4=(t+1)*UNIT_SIZE;
126                        float y4=FLOOR_Y+WALL_HEIGHT;
127                        float z4=(k+1)*UNIT_SIZE;
128
129                        float x3=(t+1)*UNIT_SIZE;
130                        float y3=FLOOR_Y;
131                        float z3=(k+1)*UNIT_SIZE;
132                        //建造三角形
133                        alVertex.add(x1);alVertex.add(y1);alVertex
                          .add(z1);
134                        alVertex.add(x2);alVertex.add(y2);alVertex
                          .add(z2);
135                        alVertex.add(x3);alVertex.add(y3);alVertex
                          .add(z3);
136
137                        alVertex.add(x3);alVertex.add(y3);alVertex
                          .add(z3);
138                        alVertex.add(x4);alVertex.add(y4);alVertex
                          .add(z4);
139                        alVertex.add(x1);alVertex.add(y1);alVertex
                          .add(z1);
140                        //建造纹理
141                        alTexture.add((float)((float)(t-area[0][1])
                           /cols));alTexture.add(0f);
142                        alTexture.add((float)((float)(t-area[0][1])
                           /cols));alTexture.add((float)((float)1
                          /rows));
```

```
143                         alTexture.add((float)((float)(t+1-area[0][1])
                          /cols));alTexture.add((float)((float)1
                          /rows));
144
145                         alTexture.add((float)((float)(t+1-area[0][1])
                          /cols));alTexture.add((float)((float)1
                          /rows));
146                         alTexture.add((float)((float)(t+1-area[0][1])
                          /cols));alTexture.add(0f);
147                        alTexture.add((float)((float)(t-area[0][1])
                          /cols));alTexture.add((float)1/rows);
148                        //建造向量
149                        alNormal.add(0f);alNormal.add(0f);
                           alNormal.add(1f);
150                        alNormal.add(0f);alNormal.add(0f);
                           alNormal.add(1f);
151                        alNormal.add(0f);alNormal.add(0f);
                           alNormal.add(1f);
152
153                        alNormal.add(0f);alNormal.add(0f);
                           alNormal.add(1f);
154                        alNormal.add(0f);alNormal.add(0f);
                           alNormal.add(1f);
155                        alNormal.add(0f);alNormal.add(0f);
                           alNormal.add(1f);
156                    }
157                    //建造墙的左面
158                    if(t==0||MAP[k][t-1]==0)
159                    {
160                        float x2=t*UNIT_SIZE;
161                        float y2=FLOOR_Y;
162                        float z2=(k+1)*UNIT_SIZE;
163
164                        float x3=t*UNIT_SIZE;
165                        float y3=FLOOR_Y+WALL_HEIGHT;
166                        float z3=(k+1)*UNIT_SIZE;
167
168                        float x4=t*UNIT_SIZE;
169                        float y4=FLOOR_Y+WALL_HEIGHT;
170                        float z4=k*UNIT_SIZE;
171
172                        float x1=t*UNIT_SIZE;
173                        float y1=FLOOR_Y;
174                        float z1=k*UNIT_SIZE;
175                        //建造三角形
176                        alVertex.add(x1);alVertex.add(y1);
                           alVertex.add(z1);
177                        alVertex.add(x2);alVertex.add(y2);
                           alVertex.add(z2);
178                        alVertex.add(x3);alVertex.add(y3);
                           alVertex.add(z3);
179
180                        alVertex.add(x3);alVertex.add(y3);
                           alVertex.add(z3);
181                        alVertex.add(x4);alVertex.add(y4);
                           alVertex.add(z4);
182                        alVertex.add(x1);alVertex.add(y1);
                           alVertex.add(z1);
183                        //建造纹理
184                        alTexture.add(0f);alTexture.add((float)
```

```
                        (k-area[0][0])/rows);
185                     alTexture.add(0f);alTexture.add((float)
                        ((float)(k+1-area[0][0])/rows));
186                     alTexture.add((float)((float)1/cols));
                        alTexture.add((float)((float)(k+1-area[0][0])
                        /rows));
187
188                     alTexture.add((float)((float)1/cols));
                        alTexture.add((float)((float)
                        (k+1-area[0][0])/rows));
189                     alTexture.add((float)((float)1/cols));
                        alTexture.add((float)(k-area[0][0])/rows);
190                     alTexture.add(0f);alTexture.add((float)
                        (k-area[0][0])/rows);
191                     //建造向量
192                     alNormal.add(-1f);alNormal.add(0f);
                        alNormal.add(0f);
193                     alNormal.add(-1f);alNormal.add(0f);
                        alNormal.add(0f);
194                     alNormal.add(-1f);alNormal.add(0f);
                        alNormal.add(0f);
195
196                     alNormal.add(-1f);alNormal.add(0f);
                        alNormal.add(0f);
197                     alNormal.add(-1f);alNormal.add(0f);
                        alNormal.add(0f);
198                     alNormal.add(-1f);alNormal.add(0f);
                        alNormal.add(0f);
199                 }
200                 //建造墙的右面
201                 if(t==cols-1||MAP[k][t+1]==0)
202                 {
203                     float x3=(t+1)*UNIT_SIZE;
204                     float y3=FLOOR_Y;
205                     float z3=(k+1)*UNIT_SIZE;
206
207                     float x2=(t+1)*UNIT_SIZE;
208                     float y2=FLOOR_Y+WALL_HEIGHT;
209                     float z2=(k+1)*UNIT_SIZE;
210
211                     float x1=(t+1)*UNIT_SIZE;
212                     float y1=FLOOR_Y+WALL_HEIGHT;
213                     float z1=k*UNIT_SIZE;
214
215                     float x4=(t+1)*UNIT_SIZE;
216                     float y4=FLOOR_Y;
217                     float z4=k*UNIT_SIZE;
218                     //建造三角形
219                     alVertex.add(x1);alVertex.add(y1);
                        alVertex.add(z1);
220                     alVertex.add(x2);alVertex.add(y2);
                        alVertex.add(z2);
221                     alVertex.add(x3);alVertex.add(y3);
                        alVertex.add(z3);
222
223                     alVertex.add(x3);alVertex.add(y3);
                        alVertex.add(z3);
224                     alVertex.add(x4);alVertex.add(y4);
                        alVertex.add(z4);
225                     alVertex.add(x1);alVertex.add(y1);
```

```
                          alVertex.add(z1);
226                       //建造纹理
227                       alTexture.add(0f);alTexture.
                          add((float)(k-area[0][0])/rows);
228                       alTexture.add(0f);alTexture.
                          add((float)((float)(k+1-area[0][0])/rows));
229                       alTexture.add((float)((float)1/cols));
                          alTexture.add((float)((float)(k+1-area[0][0])
                          /rows));
230
231                       alTexture.add((float)((float)1/cols));
                          alTexture.add((float)((float)(k+1-area[0][0])
                          /rows));
232                       alTexture.add((float)((float)1/cols));
                          alTexture.add((float)(k-area[0][0])/rows);
233                       alTexture.add(0f);alTexture
                          .add((float)(k-area[0][0])/rows);
234                       //建造向量
235                       alNormal.add(1f);alNormal
                          .add(0f);alNormal.add(0f);
236                       alNormal.add(1f);alNormal
                          .add(0f);alNormal.add(0f);
237                       alNormal.add(1f);alNormal
                          .add(0f);alNormal.add(0f);
238
239                       alNormal.add(1f);alNormal
                          .add(0f);alNormal.add(0f);
240                       alNormal.add(1f);alNormal
                          .add(0f);alNormal.add(0f);
241                       alNormal.add(1f);alNormal
                          .add(0f);alNormal.add(0f);
242                    }
243                }
244            }
245        }
246     }
247     }
248     vCount=alVertex.size()/3;
249     float vertices[]=new float[alVertex.size()];
250     for(int i=0;i<alVertex.size();i++)
251     {
252      vertices[i]=alVertex.get(i);
253     }
254
255     //创建顶点坐标数据缓冲
256     //vertices.length*4 是因为一个 float4 个字节
257     ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
258     //设置字节顺序
259     vbb.order(ByteOrder.nativeOrder());
260     //转换为 float 型缓冲
261     mVertexBuffer = vbb.asFloatBuffer();
262     //向缓冲区中放入顶点坐标数据
263     mVertexBuffer.put(vertices);
264     //设置缓冲区起始位置
265     mVertexBuffer.position(0);
266     //特别提示：由于不同平台字节顺序不同，数据单元不是字节的一定要经过
        ByteBuffer
267     //转换，关键是要通过 ByteOrder 设置 nativeOrder()，否则有可能会出问题
```

```
268         //顶点坐标数据的初始化——end
269
270         //顶点法向量数据初始化 ================begin====================
271         float normals[]=new float[vCount*3];
272         for(int i=0;i<vCount*3;i++)
273         {
274          normals[i]=alNormal.get(i);
275         }
276         //分配空间
277         ByteBuffer nbb = ByteBuffer.allocateDirect(normals.length*4);
278         //设置字节顺序
279         nbb.order(ByteOrder.nativeOrder());
280         //转换为 float 型缓冲
281         mNormalBuffer = nbb.asFloatBuffer();
282         //向缓冲区中放入顶点法向量数据
283         mNormalBuffer.put(normals);
284         //设置缓冲区起始位置
285         mNormalBuffer.position(0);
286         //顶点法向量数据初始化——end
287
288         //顶点纹理数据的初始化——begin
289         float textures[]=new float[alTexture.size()];
290         for(int i=0;i<alTexture.size();i++)
291         {
292          textures[i]=alTexture.get(i);
293         }
294         //创建顶点纹理数据缓冲
295         ByteBuffer tbb = ByteBuffer.allocateDirect(textures.length*4);
296         //设置字节顺序
297         tbb.order(ByteOrder.nativeOrder());
298         //转换为 Float 型缓冲
299         mTextureBuffer= tbb.asFloatBuffer();
300         //向缓冲区中放入顶点着色数据
301         mTextureBuffer.put(textures);
302         //设置缓冲区起始位置
303         mTextureBuffer.position(0);
304         //特别提示：由于不同平台字节顺序不同，数据单元不是字节的一定要经过
            ByteBuffer
305         //转换，关键是要通过 ByteOrder 设置 nativeOrder()，否则有可能会出问题
306         //顶点纹理数据的初始化——end
307     }
```

（5）上面的步骤可以看作是对地图数据的解析和初始化，最后在需要绘图的时候直接调用 drawself 就可以绘制出墙壁。在绘制函数为画笔指定坐标数据、法向量数据、纹理数据之后，调用 gl.glDrawArrays 即可完成绘制。

```
01  public void drawSelf(GL10 gl,int texId)
02      {
03          //为画笔指定顶点坐标数据
04          gl.glVertexPointer
05          (
06                  3,                  //每个顶点的坐标由 xyz 组成
07                  GL10.GL_FLOAT,      //顶点坐标值的类型为 GL_FIXED
08                  0,                  //连续顶点坐标数据之间的间隔
09                  mVertexBuffer       //顶点坐标数据
10          );
```

```
11
12          //为画笔指定顶点法向量数据
13          gl.glNormalPointer(GL10.GL_FLOAT, 0, mNormalBuffer);
14          //为画笔指定纹理 ST 坐标缓冲
15          gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTextureBuffer);
16          //绑定当前纹理
17          gl.glBindTexture(GL10.GL_TEXTURE_2D, texId);
18
19          //绘制图形
20          gl.glDrawArrays
21          (
22                GL10.GL_TRIANGLES,      //以三角形方式填充
23                0,
24                vCount
25          );
26       }
```

22.2.4　小球

如图 22.5 所示，我们将小球从上到下切割成一个个“薄片”，这个薄片可以近似为一个圆柱，接着我们又将这个薄片沿着半径切割，切割的表面可以近似为一个正方形。再将该正方形切割成两个等腰三角形，即可得到 6 个顶点，就这样我们将一个球切割成一个个正方形，最后贴上纹理就成了一个立体的小球。

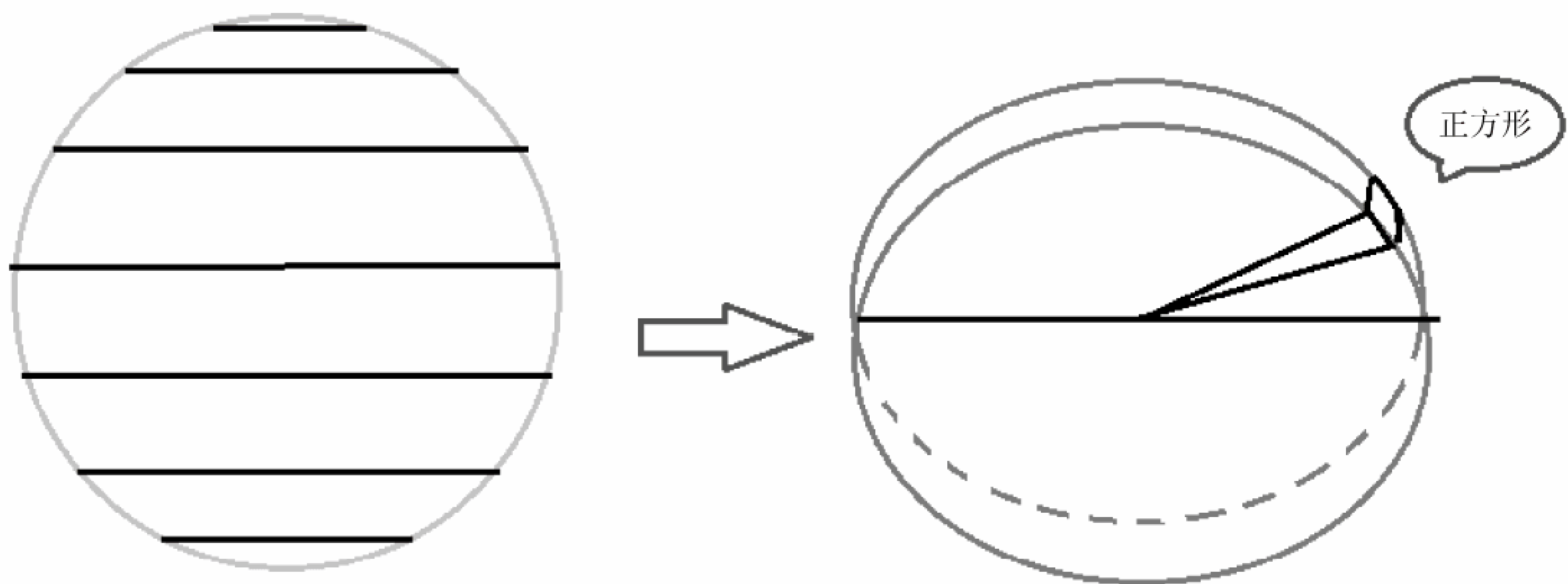


图 22.5　小球切割方法

我们新建小球类 BallTextureByVertext.java，首先实现纹理切割函数 generateTexCoor，如下所示，根据传入的参数决定要切割的行数、列数，将切割得到的每一个矩形分割成两个三角形，并将三角形的顶点存储到数组中，为我们后面绘制球表面纹理作准备。

```
01  //自动切分纹理产生纹理数组的方法
02    public float[] generateTexCoor(int bw,int bh)//传入切分的列数，行数
03    {
04      float[] result=new float[bw*bh*6*2];
05      float sizew=1.0f/bw;          //列宽
06      float sizeh=1.0f/bh;          //行宽
07      int c=0;
```

```
08      for(int i=0;i<bh;i++)
09      {
10          for(int j=0;j<bw;j++)
11          {
12              //每一个矩形，由两个三角形构成，共 6 个点、12 个纹理坐标
13              float s=j*sizew;
14              float t=i*sizeh;
15
16              result[c++]=s;
17              result[c++]=t;
18
19              result[c++]=s;
20              result[c++]=t+sizeh;
21
22              result[c++]=s+sizew;
23              result[c++]=t;
24
25
26              result[c++]=s+sizew;
27              result[c++]=t;
28
29              result[c++]=s;
30              result[c++]=t+sizeh;
31
32              result[c++]=s+sizew;
33              result[c++]=t+sizeh;
34          }
35      }
36      return result;
37    }
```

在类的开始依旧声明顶点坐标数据缓冲、顶点法向量数据缓冲、顶点纹理数据缓冲这 3 个变量，用于存储小球的坐标数据。根据我们上面的分析，我们将小球切割成一个个柱面，并将柱面沿着圆周切割成一个个矩形，将这个矩形的 4 个顶点计算出来。最后将该矩形分割成两个三角形进行存储，并根据顶点的行列获取对应的纹理。因为顶点的坐标刚好等于法向量坐标，因此直接将顶点坐标作为法向量坐标存储即可。

```
001 //小球绘制类
002 public class BallTextureByVertex {
003     private FloatBuffer   mVertexBuffer;            //顶点坐标数据缓冲
004     private FloatBuffer   mNormalBuffer;            //顶点法向量数据缓冲
005     private FloatBuffer mTextureBuffer;             //顶点纹理数据缓冲
006     public float mAngleX;                           //沿 x 轴旋转角度
007     public float mAngleY;                           //沿 y 轴旋转角度
008     public float mAngleZ;                           //沿 z 轴旋转角度
009     int vCount=0;                                   //顶点数量
010     public BallTextureByVertex(float scale,float angleSpan)
011     {
012         //获取切分整图的纹理数组
013         float[] texCoorArray=
014          generateTexCoor
015          (
016                 (int)(360/angleSpan),               //纹理图切分的列数
017                 (int)(180/angleSpan)                //纹理图切分的行数
018         );
019         int tc=0;                                   //纹理数组计数器
020         int ts=texCoorArray.length;                 //纹理数组长度
```

```
021
022         ArrayList<Float> alVertix=new ArrayList<Float>();
                                        //存放顶点坐标的 ArrayList
023         ArrayList<Float> alTexture=new ArrayList<Float>();
                                        //存放纹理坐标的 ArrayList
024
025         for(float vAngle=90;vAngle>-90;vAngle=vAngle-angleSpan)
                                        //垂直方向每一份为 angleSpan 度
026         {
027             for(float hAngle=360;hAngle>0;hAngle=hAngle-angleSpan)
                                        //水平方向每一份为 angleSpan 度
028             {
029                 //纵向、横向各到一个角度后，计算对应的此点在球面上的四边形顶点坐标
030                 //并构建两个组成四边形的三角形
031                 doublexozLength=scale*UNIT_SIZE*Math.cos
                    (Math.toRadians(vAngle));
032                 float x1=(float)(xozLength*Math.cos
                   (Math.toRadians(hAngle)));
033                 float z1=(float)(xozLength*Math.sin
                   (Math.toRadians(hAngle)));
034                 float y1=(float)(scale*UNIT_SIZE*Math.sin
                    (Math.toRadians(vAngle)));
035
036                 xozLength=scale*UNIT_SIZE*Math.cos
                    (Math.toRadians(vAngle-angleSpan));
037                 float x2=(float)(xozLength*Math.cos
                    (Math.toRadians(hAngle)));
038                 float z2=(float)(xozLength*Math.sin
                   (Math.toRadians(hAngle)));
039                 float y2=(float)(scale*UNIT_SIZE*Math.sin
                    (Math.toRadians(vAngle-angleSpan)));
040
041                 xozLength=scale*UNIT_SIZE*Math.cos
                    (Math.toRadians(vAngle-angleSpan));
042                 float x3=(float)(xozLength*Math.cos
                    (Math.toRadians(hAngle-angleSpan)));
043                 float z3=(float)(xozLength*Math.sin
                    Math.toRadians(hAngle-angleSpan)));
044                 float y3=(float)(scale*UNIT_SIZE*Math.sin
                    (Math.toRadians(vAngle-angleSpan)));
045
046                 xozLength=scale*UNIT_SIZE*Math.cos
                    (Math.toRadians(vAngle));
047                 float x4=(float)(xozLength*Math.cos
                    (Math.toRadians(hAngle-angleSpan)));
048                 float z4=(float)(xozLength*Math.sin
                    (Math.toRadians(hAngle-angleSpan)));
049                 float y4=(float)(scale*UNIT_SIZE*Math.sin
                    (Math.toRadians(vAngle)));
050
051                 //构建第一个三角形
052                 alVertix.add(x1);alVertix.add(y1);alVertix.add(z1);
053                 alVertix.add(x2);alVertix.add(y2);alVertix.add(z2);
054                 alVertix.add(x4);alVertix.add(y4);alVertix.add(z4);
055                 //构建第二个三角形
056                 alVertix.add(x4);alVertix.add(y4);alVertix.add(z4);
057                 alVertix.add(x2);alVertix.add(y2);alVertix.add(z2);
058                 alVertix.add(x3);alVertix.add(y3);alVertix.add(z3);
059
```

```
060                 //第一个三角形 3 个顶点的 6 个纹理坐标
061                 alTexture.add(texCoorArray[tc++%ts]);
062                 alTexture.add(texCoorArray[tc++%ts]);
063                 alTexture.add(texCoorArray[tc++%ts]);
064                 alTexture.add(texCoorArray[tc++%ts]);
065                 alTexture.add(texCoorArray[tc++%ts]);
066                 alTexture.add(texCoorArray[tc++%ts]);
067                 //第二个三角形 3 个顶点的 6 个纹理坐标
068                 alTexture.add(texCoorArray[tc++%ts]);
069                 alTexture.add(texCoorArray[tc++%ts]);
070                 alTexture.add(texCoorArray[tc++%ts]);
071                 alTexture.add(texCoorArray[tc++%ts]);
072                 alTexture.add(texCoorArray[tc++%ts]);
073                 alTexture.add(texCoorArray[tc++%ts]);
074             }
075         }
076
077
078
079         vCount=alVertix.size()/3;
          //顶点的数量为坐标值数量的 1/3，因为一个顶点有 3 个坐标
080
081         //将 alVertix 中的坐标值转存到一个 float 数组中
082         float vertices[]=new float[vCount*3];
083         for(int i=0;i<alVertix.size();i++)
084         {
085             vertices[i]=alVertix.get(i);
086         }
087
088         //创建绘制顶点数据缓冲
089         ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
090         vbb.order(ByteOrder.nativeOrder());           //设置字节顺序
091         mVertexBuffer = vbb.asFloatBuffer();          //转换为 int 型缓冲
092         mVertexBuffer.put(vertices);          //向缓冲区中放入顶点坐标数据
093         mVertexBuffer.position(0);            //设置缓冲区起始位置
094
095         //创建顶点法向量数据缓冲
096         ByteBuffer nbb = ByteBuffer.allocateDirect(vertices.length*4);
097         nbb.order(ByteOrder.nativeOrder());           //设置字节顺序
098         mNormalBuffer = vbb.asFloatBuffer();          //转换为 int 型缓冲
099         mNormalBuffer.put(vertices);          //向缓冲区中放入顶点坐标数据
100         mNormalBuffer.position(0);            //设置缓冲区起始位置
101
102         //创建纹理坐标缓冲
103         float textureCoors[]=new float[alTexture.size()];
         //顶点纹理值数组
104         for(int i=0;i<alTexture.size();i++)
105         {
106             textureCoors[i]=alTexture.get(i);
107         }
108
109         ByteBuffer tbb = ByteBuffer.allocateDirect(textureCoors.
            length*4);
```

```
110         tbb.order(ByteOrder.nativeOrder());          //设置字节顺序
111         mTextureBuffer = tbb.asFloatBuffer();        //转换为 int 型缓冲
112         mTextureBuffer.put(textureCoors);    //向缓冲区中放入顶点着色数据
113         mTextureBuffer.position(0);          //设置缓冲区起始位置
114     }
```

最后实现绘制函数 drawSelf()，考虑到小球转动的时候方向是千变万化的，因此先将小球依次沿 X、Y、Z 轴旋转设置好的角度值，再根据之前数组中的数据将小球绘制出来。

```
01  //绘制小球
02    public void drawSelf(GL10 gl,int texId)
03    {
04        gl.glRotatef(mAngleZ, 0, 0, 1);                    //沿 Z 轴旋转
05        gl.glRotatef(mAngleX, 1, 0, 0);                    //沿 X 轴旋转
06        gl.glRotatef(mAngleY, 0, 1, 0);                    //沿 Y 轴旋转
07
08        //允许使用顶点数组
09        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
10       //为画笔指定顶点坐标数据
11        gl.glVertexPointer
12        (
13              3,                          //每个顶点的坐标由 xyz 组成
14              GL10.GL_FLOAT,              //顶点坐标值的类型为 GL_FIXED
15              0,                  //连续顶点坐标数据之间的间隔
16              mVertexBuffer       //顶点坐标数据
17        );
18
19
20        //为画笔指定顶点法向量数据
21        gl.glNormalPointer(GL10.GL_FLOAT, 0, mNormalBuffer);
22        //为画笔指定纹理 ST 坐标缓冲
23        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTextureBuffer);
24        //绑定当前纹理
25        gl.glBindTexture(GL10.GL_TEXTURE_2D, texId);
26
27        //绘制图形
28        gl.glDrawArrays
29        (
30              GL10.GL_TRIANGLES,      //以三角形方式填充
31              0,                      //开始点编号
32              vCount                  //顶点数量
33        );
34    }
```

22.2.5 圆形洞

如下所示，圆形洞的绘制方法和地板类似，直接将准备好的圆形洞图片完整地绘制到地图上即可。

```
01  //圆形洞
02  public class RectWall
03  {
```

```
04      private FloatBuffer   mVertexBuffer;                //顶点坐标数据缓冲
05      private FloatBuffer   mTextureBuffer;               //顶点纹理数据缓冲
06      int vCount;                                         //顶点数
07      float x;
08      float y;
09      float z;
10      public RectWall(float width,float height)
11      {
12          vCount=6;
13          float []vertices=new float[]
14          {
15                  -width*UNIT_SIZE/2,height*UNIT_SIZE/2,0,
16                  -width*UNIT_SIZE/2,-height*UNIT_SIZE/2,0,
17                  width*UNIT_SIZE/2,-height*UNIT_SIZE/2,0,
18
19                  width*UNIT_SIZE/2,-height*UNIT_SIZE/2,0,
20                  width*UNIT_SIZE/2,height*UNIT_SIZE/2,0,
21                  -width*UNIT_SIZE/2,height*UNIT_SIZE/2,0
22          };
23          ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
24          vbb.order(ByteOrder.nativeOrder());             //设置字节顺序
25          mVertexBuffer = vbb.asFloatBuffer();            //转换为 float 型缓冲
26          mVertexBuffer.put(vertices);            //向缓冲区中放入顶点坐标数据
27          mVertexBuffer.position(0);                      //设置缓冲区起始位置
28          float textures[]=new float[]
29          {
30                  0,0,
31                  0,1f,
32                  1f,1f,
33
34                  1f,1f,
35                  1,0,
36                  0,0
37          };
38
39          ByteBuffer tbb = ByteBuffer.allocateDirect(textures.length*4);
40          tbb.order(ByteOrder.nativeOrder());                 //设置字节顺序
41          mTextureBuffer= tbb.asFloatBuffer();        //转换为 float 型缓冲
42          mTextureBuffer.put(textures);           //向缓冲区中放入顶点着色数据
43          mTextureBuffer.position(0);                 //设置缓冲区起始位置
44      }
45
46       public void drawSelf(GL10 gl,int texId)
47          {
48
49              gl.glPushMatrix();
50              gl.glTranslatef(x, y, z);
51            //为画笔指定顶点坐标数据
52              gl.glVertexPointer
53              (
54                      3,                      //每个顶点的坐标由 xyz 组成
55                      GL10.GL_FLOAT,          //顶点坐标值的类型为 GL_FIXED
```

```
56                  0,                          //连续顶点坐标数据之间的间隔
57                  mVertexBuffer               //顶点坐标数据
58              );
59
60
61              //为画笔指定纹理 ST 坐标缓冲
62              gl.glTexCoordPointer(2, GL10.GL FLOAT, 0, mTextureBuffer);
63              //绑定当前纹理
64              gl.glBindTexture(GL10.GL TEXTURE 2D, texId);
65
66              //绘制图形
67              gl.glDrawArrays
68              (
69                  GL10.GL_TRIANGLES,      //以三角形方式填充
70                  0,
71                  vCount
72              );
73              gl.glPopMatrix();
74          }
75  }
```

22.2.6　数字

绘制数字与绘制地板和绘制圆形洞是一个道理，在该类中变量 NumberStr 用于存放将要绘制的数字。需要注意的是，绘制分钟与绘制秒钟的顺序刚好相反，绘制秒钟按照正常的顺序是从左往右绘制，而绘制分钟需要从分钟的末位开始向首位绘制。

```
01  //绘制数字的类
02  public class Number
03  {
04      GameSurfaceView mv;
05      TextureRect[] numbers=new TextureRect[10];
06      String NumberStr;
07      public float y;
08      public Number(GameSurfaceView mv)
09      {
10          this.mv=mv;
11
12          //生成 0~9 十个数字的纹理矩形
13          for(int i=0;i<10;i++)
14          {
15              numbers[i]=new TextureRect
16              (
17                  ICON_WIDTH*0.7f/2,
18                  ICON_HEIGHT*0.7f/2,
19                  new float[]
20                  {
21                      0.1f*i,0, 0.1f*i,1, 0.1f*(i+1),0,
22                      0.1f*i,1, 0.1f*(i+1),1,  0.1f*(i+1),0
23                  }
24              );
25          }
26      }
27
28      public void drawSelf(GL10 gl,int flag,int texId)
29      {
```

```
30          for(int i=0;i<NumberStr.length();i++)
31          {//将得分中的每个数字字符绘制
32              char c=NumberStr.charAt(flag==1?i:NumberStr.length()-i-1);
33              //保存当前状态
34              gl.glPushMatrix();
35              gl.glTranslatef(flag==1?i*ICON_WIDTH*
                0.7f:-i*ICON_WIDTH*0.7f,y, 0);
36              numbers[c-'0'].drawSelf(gl,texId);
37              //恢复成原来的状态
38              gl.glPopMatrix();
39          }
40      }
41  }
```

绘制时间的时候使用了类 TextureRect，用于绘制矩形图片，该类的功能比较简单，根据提供的纹理、高宽信息绘制矩形图片。

```
01  //绘制矩形图片类
02  public class TextureRect
03  {
04      private FloatBuffer   mVertexBuffer;              //顶点坐标数据缓冲
05      private FloatBuffer   mTextureBuffer;             //顶点着色数据缓冲
06      int vCount;
07      public TextureRect(float X_UNIT_SIZE,float Y_UNIT_SIZE,float[]
        textures)
08      {
09          //顶点坐标数据的初始化——begin
10          vCount=6;
11          float vertices[]=new float[]
12          {
13              -1*X_UNIT_SIZE,1*Y_UNIT_SIZE,0,
14              -1*X_UNIT_SIZE,-1*Y_UNIT_SIZE,0,
15              1*X_UNIT_SIZE,1*Y_UNIT_SIZE,0,
16
17              -1*X_UNIT_SIZE,-1*Y_UNIT_SIZE,0,
18              1*X_UNIT_SIZE,-1*Y_UNIT_SIZE,0,
19              1*X_UNIT_SIZE,1*Y_UNIT_SIZE,0
20          };
21
22          //创建顶点坐标数据缓冲
23          //vertices.length*4 是因为一个整数 4 字节
24          ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
25          vbb.order(ByteOrder.nativeOrder());           //设置字节顺序
26          mVertexBuffer = vbb.asFloatBuffer();          //转换为 int 型缓冲
27          mVertexBuffer.put(vertices);      //向缓冲区中放入顶点坐标数据
28          mVertexBuffer.position(0);        //设置缓冲区起始位置
29          //特别提示：由于不同平台字节顺序不同，据单元不是字节的一定要经过
            ByteBuffer
30          //转换，关键是要通过 ByteOrder 设置 nativeOrder()，否则有可能会出问题
31          //顶点坐标数据的初始化——end
32
33          //顶点纹理数据的初始化——begin
34          //创建顶点纹理数据缓冲
35          ByteBuffer tbb = ByteBuffer.allocateDirect(textures.length*4);
36          tbb.order(ByteOrder.nativeOrder());           //设置字节顺序
```

```
37          mTextureBuffer= tbb.asFloatBuffer();     //转换为 Float 型缓冲
38          mTextureBuffer.put(textures);        //向缓冲区中放入顶点着色数据
39          mTextureBuffer.position(0);                 //设置缓冲区起始位置
40          //特别提示：由于不同平台字节顺序不同，据单元不是字节的一定要经过
            ByteBuffer
41          //转换，关键是要通过 ByteOrder 设置 nativeOrder()，否则有可能会出问题
42          //顶点纹理数据的初始化——end
43      }
44      //绘制
45      public void drawSelf(GL10 gl,int texId)
46      {
47          gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);//启用顶点坐标数组
48
49          //为画笔指定顶点坐标数据
50          gl.glVertexPointer
51          (
52                  3,                          //每个顶点的坐标由 xyz 组成
53                  GL10.GL_FLOAT,              //顶点坐标值的类型为 GL_FIXED
54                  0,                          //连续顶点坐标数据之间的间隔
55                  mVertexBuffer               //顶点坐标数据
56          );
57          //为画笔指定纹理 ST 坐标缓冲
58          gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTextureBuffer);
59          //绑定当前纹理
60          gl.glBindTexture(GL10.GL TEXTURE 2D, texId);
61          //绘制图形
62          gl.glDrawArrays
63          (
64                  GL10.GL_TRIANGLES,      //以三角形方式填充
65                  0,
66                  vCount
67          );
68      }
69  }
```

22.3　游 戏 菜 单

如图 22.6 所示为游戏菜单，当我们单击游戏图标进入游戏时，首先出现在我们眼前的就是这个游戏菜单，从游戏菜单中我们可以选择开始游戏、进入排行榜、设置游戏。



图 22.6　游戏菜单

22.3.1　界面布局

（1）游戏菜单的界面布局如下所示，从上到下有 3 个 ImageButton，分别为“开始”、“排行榜”、“设置”。

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
03     android:orientation="vertical"
04     android:layout_width="fill_parent"
05     android:layout_height="fill_parent"
06     android:gravity="center_horizontal"
07     android:background="@drawable/main">
08     <!-- 开始 -->
09     <ImageButton
10        android:id="@+id/ImageButton_Start"
11        android:background="@drawable/start_button"
12        android:layout_marginTop="90dp"
13        android:layout_width="wrap_content"
14        android:layout_height="wrap_content" />
15     <!-- 排行榜 -->
16     <ImageButton
17        android:id="@+id/ImageButton_Rank"
18        android:background="@drawable/rank_button"
19        android:layout_marginTop="10dp"
20        android:layout_width="wrap_content"
21        android:layout_height="wrap_content" />
22     <!-- 设置 -->
23     <ImageButton
24        android:id="@+id/ImageButton_Set"
25        android:background="@drawable/set_button"
26        android:layout_marginTop="10dp"
27        android:layout_width="wrap_content"
28        android:layout_height="wrap_content" />
29  </LinearLayout>
```

（2）单击“开始”按钮之后将进入游戏选关界面，如图 22.7 所示，共有 6 个 ImageView 组成的按钮，每 3 个按钮为一排，分别是第 1 关到第 6 关，单击相应的按钮可以进入对应的关卡。

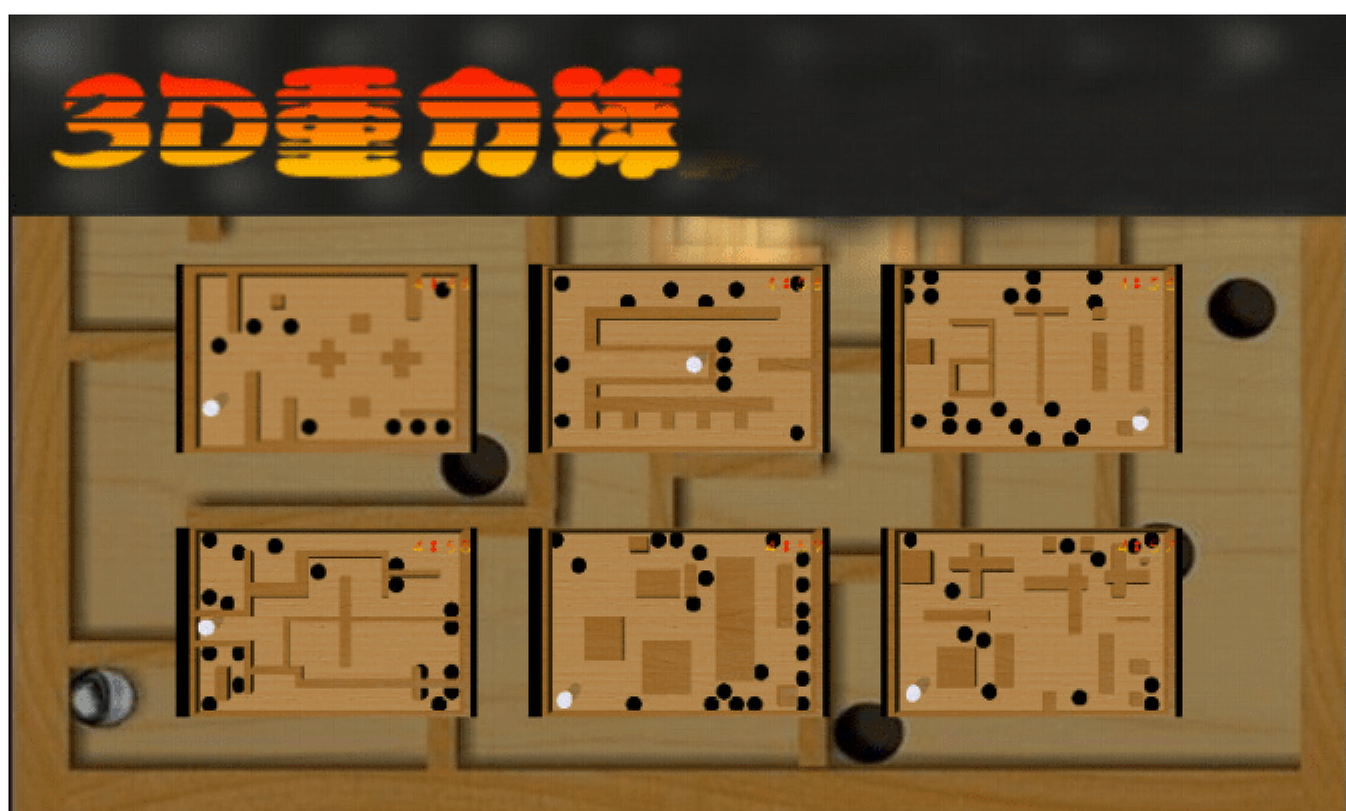


图 22.7　游戏选关界面

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android
    .com/apk/res/android"
03      android:orientation="vertical"
04      android:layout_width="fill_parent"
05      android:layout_height="fill_parent"
06      android:gravity="center_horizontal"
07      android:background="@drawable/main">
08      <!-- 第 1 排 -->
09      <LinearLayout
10          android:orientation="horizontal"
11          android:layout_marginTop="100dp"
12          android:layout_marginLeft="10dp"
13          android:layout_marginRight="10dp"
14          android:layout_marginBottom="10dp"
15          android:gravity="center"
16          android:layout_width="wrap_content"
17          android:layout_height="wrap_content">
18          <!-- 第 1 关 -->
19          <ImageButton
20              android:id="@+id/ImageButton_map01"
21              android:background="@drawable/map01_button"
22              android:layout_marginRight="20dp"
23              android:layout_width="120dp"
24              android:layout_height="75dp" />
25          <!-- 第 2 关 -->
26          <ImageButton
27              android:id="@+id/ImageButton_map02"
28              android:background="@drawable/map02_button"
29              android:layout_marginRight="20dp"
30              android:layout_width="120dp"
31              android:layout_height="75dp" />
32          <!-- 第 3 关 -->
33          <ImageButton
34              android:id="@+id/ImageButton_map03"
35              android:background="@drawable/map03_button"
36              android:layout_width="120dp"
37              android:layout_height="75dp" />
38          </LinearLayout>
39      <!-- 第 2 排 -->
40      <LinearLayout
41          android:orientation="horizontal"
42          android:layout_marginTop="20dp"
43          android:layout_marginLeft="10dp"
44          android:layout_marginRight="10dp"
45          android:layout_marginBottom="10dp"
46          android:gravity="center"
47          android:layout_width="wrap_content"
48          android:layout_height="wrap_content">
49          <!-- 第 4 关 -->
50          <ImageButton
51              android:id="@+id/ImageButton_map04"
52              android:layout_marginRight="20dp"
53              android:background="@drawable/map04_button"
54              android:layout_width="120dp"
55              android:layout_height="75dp" />
56          <!-- 第 5 关 -->
57          <ImageButton
58              android:id="@+id/ImageButton_map05"
59              android:layout_marginRight="20dp"
```

```
60              android:background="@drawable/map05_button"
61              android:layout_width="120dp"
62              android:layout_height="75dp" />
63          <!-- 第 6 关 -->
64          <ImageButton
65              android:id="@+id/ImageButton_map06"
66              android:background="@drawable/map06_button"
67              android:layout_width="120dp"
68              android:layout_height="75dp" />
69          </LinearLayout>
70  </LinearLayout>
```

（3）单击“排行榜”按钮可以进入分数查看界面，如图 22.7 所示，整体采用从上到下的布局，最上面有两个加、减按钮，用来选择关数。中间显示标题、成绩和时间，下面从上到下依次显示第 1 名到第 5 名的成绩。



图 22.8　排行榜界面

```
001 <?xml version="1.0" encoding="utf-8"?>
002 <LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
003     android:orientation="vertical"
004     android:layout_width="fill_parent"
005     android:layout_height="fill_parent"
006     android:gravity="center_horizontal"
007     android:background="@drawable/main">
008         <LinearLayout
009             android:orientation="horizontal"
010             android:layout_marginTop="90dp"
011             android:gravity="center_horizontal"
012             android:layout_width="wrap_content"
013             android:layout_height="wrap_content">
014             <!-- 向左选关 -->
015             <ImageButton
016                 android:id="@+id/ImageButton_Left"
017                 android:background="@drawable/left_button"
018                 android:layout_marginLeft="10dp"
019                 android:layout_marginRight="20dp"
020                 android:layout_width="50dp"
021                 android:layout_height="25dp" />
022             <!-- 当前查看的关数 -->
023             <TextView
024                 android:text="第一关"
025                 android:id="@+id/TextView_level"
```

```
026                 android:textSize="20sp"
027                 android:textColor="#ff0000"
028                 android:layout width="wrap content"
029                 android:layout height="wrap content" />
030             <!-- 向右选关 -->
031             <ImageButton
032                 android:id="@+id/ImageButton_Right"
033                 android:background="@drawable/right_button"
034                 android:layout_marginLeft="20dp"
035                 android:layout_marginRight="10dp"
036                 android:layout_width="50dp"
037                 android:layout height="25dp" />
038         </LinearLayout>
039         <LinearLayout
040             android:orientation="horizontal"
041             android:layout width="wrap content"
042             android:gravity="center horizontal"
043             android:layout height="wrap content">
044             <!-- 成绩 -->
045             <TextView
046                 android:text="成绩"
047                 android:textColor="#ff0000"
048                 android:textSize="18sp"
049                 android:layout marginRight="40dp"
050                 android:layout width="wrap content"
051                 android:layout height="wrap content" />
052             <!-- 游戏时间 -->
053             <TextView
054                 android:text="游戏时间"
055                 android:textSize="18sp"
056                 android:textColor="#ff0000"
057                 android:layout marginLeft="40dp"
058                 android:layout width="wrap content"
059                 android:layout height="wrap content" />
060         </LinearLayout>
061         <LinearLayout
062             android:orientation="vertical"
063             android:layout marginTop="5dp"
064             android:gravity="center horizontal"
065             android:layout_width="wrap_content"
066             android:layout height="wrap content">
067             <!-- 第 1 名 -->
068             <LinearLayout
069                 android:orientation="horizontal"
070                 android:layout marginTop="10dp"
071                 android:gravity="center horizontal"
072                 android:layout width="wrap content"
073                 android:layout height="wrap content">
074                 <!-- 成绩 -->
075                 <TextView
076                     android:id="@+id/TextView_01"
077                     android:textColor="#f3ea32"
078                     android:textSize="16sp"
079                     android:layout_marginRight="40dp"
080                     android:layout_width="wrap_content"
081                     android:layout_height="wrap_content" />
082                 <!-- 游戏时间 -->
083                 <TextView
084                     android:id="@+id/TextView 11"
085                     android:textColor="#f3ea32"
```

```
086                 android:textSize="16sp"
087                 android:layout marginLeft="40dp"
088                 android:layout width="wrap content"
089                 android:layout height="wrap content" />
090         </LinearLayout>
091         <!-- 第 2 名 -->
092         <LinearLayout
093             android:orientation="horizontal"
094             android:layout_marginTop="10dp"
095             android:layout_width="wrap_content"
096             android:layout_height="wrap_content">
097             <!-- 成绩 -->
098             <TextView
099                 android:id="@+id/TextView 02"
100                 android:textColor="#f3ea32"
101                 android:textSize="16sp"
102                 android:layout marginRight="40dp"
103                 android:layout width="wrap content"
104                 android:layout height="wrap content" />
105             <!-- 游戏时间 -->
106             <TextView
107                 android:id="@+id/TextView 22"
108                 android:textColor="#f3ea32"
109                 android:textSize="16sp"
110                 android:layout marginLeft="40dp"
111                 android:layout width="wrap content"
112                 android:layout height="wrap content" />
113         </LinearLayout>
114         <!-- 第 3 名 -->
115         <LinearLayout
116             android:orientation="horizontal"
117             android:layout_marginTop="10dp"
118             android:layout_width="wrap_content"
119             android:layout_height="wrap_content">
120             <!-- 成绩 -->
121             <TextView
122                 android:id="@+id/TextView 03"
123                 android:textColor="#f3ea32"
124                 android:textSize="16sp"
125                 android:layout_marginRight="40dp"
126                 android:layout width="wrap content"
127                 android:layout height="wrap content" />
128             <!-- 游戏时间 -->
129             <TextView
130                 android:id="@+id/TextView 33"
131                 android:textColor="#f3ea32"
132                 android:textSize="16sp"
133                 android:layout marginLeft="40dp"
134                 android:layout width="wrap content"
135                 android:layout height="wrap content" />
136         </LinearLayout>
137         <!-- 第 4 名 -->
138         <LinearLayout
139             android:orientation="horizontal"
140             android:layout_marginTop="10dp"
141             android:layout_width="wrap_content"
142             android:layout_height="wrap_content">
143             <!-- 成绩 -->
144             <TextView
145                 android:id="@+id/TextView_04"
```

```
146                 android:textColor="#f3ea32"
147                 android:textSize="16sp"
148                 android:layout_marginRight="40dp"
149                 android:layout_width="wrap_content"
150                 android:layout_height="wrap_content" />
151             <!-- 游戏时间 -->
152             <TextView
153                 android:id="@+id/TextView_44"
154                 android:textColor="#f3ea32"
155                 android:textSize="16sp"
156                 android:layout_marginLeft="40dp"
157                 android:layout_width="wrap_content"
158                 android:layout_height="wrap_content" />
159         </LinearLayout>
160         <!-- 第 5 名 -->
161         <LinearLayout
162             android:orientation="horizontal"
163             android:layout_marginTop="10dp"
164             android:layout_width="wrap_content"
165             android:layout_height="wrap_content">
166             <!-- 成绩 -->
167             <TextView
168                 android:id="@+id/TextView_05"
169                 android:layout_width="wrap_content"
170                 android:textColor="#f3ea32"
171                 android:textSize="16sp"
172                 android:layout_marginRight="40dp"
173                 android:layout_height="wrap_content" />
174             <!-- 游戏时间 -->
175             <TextView
176                 android:id="@+id/TextView_55"
177                 android:textColor="#f3ea32"
178                 android:textSize="16sp"
179                 android:layout_marginLeft="40dp"
180                 android:layout_width="wrap_content"
181                 android:layout_height="wrap_content" />
182         </LinearLayout>
183     </LinearLayout>
184 </LinearLayout>
```

（4）单击“设置”按钮进入设置界面，如图 22.9 所示，上面有一个选框用于设置碰壁声音的开启，下面一个“确定”按钮用于保存设置。



图 22.9　游戏设置界面

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <LinearLayout xmlns:android="http://schemas.android.com/apk/res
    /android"
03      android:orientation="vertical"
04      android:layout width="fill parent"
05      android:layout height="fill parent"
06      android:gravity="center horizontal"
07      android:background="@drawable/main">
08      <!-- 选项 -->
09      <LinearLayout
10          android:orientation="horizontal"
11          android:layout width="wrap content"
12          android:layout marginTop="100dp"
13          android:layout_height="wrap_content">
14          <!-- 选项内容 -->
15          <TextView
16              android:text="是否开启碰壁声音"
17              android:textSize="25sp"
18              android:layout width="wrap content"
19              android:layout height="wrap content" />
20          <!-- 选项框 -->
21          <CheckBox
22              android:id="@+id/CheckBox collision"
23              android:width="35dp"
24              android:height="35dp"
25              android:layout width="wrap content"
26              android:layout_height="wrap_content" />
27      </LinearLayout>
28      <!—“确定”按钮-->
29      <ImageButton
30          android:id="@+id/ImageButton ok"
31          android:layout width="wrap content"
32          android:background="@drawable/ok button"
33          android:layout height="wrap content" />
34  </LinearLayout>
```

22.3.2　主菜单功能

（1）如下所示为主界面函数的初始化函数，游戏开始时设置游戏为全屏显示，并设置为横屏，接着取得屏幕的高宽信息保存到全局变量中供程序后面使用。接着初始化 G-Sensor，初始化游戏声音，初始化数据库，最后进入主菜单界面。

```
01  //当前枚举值
02  WhichView curr;
03  //进入游戏界面
04  GameSurfaceView msv;
05  //排行榜界面
06  GameView gameView;
07  //是否开启碰撞声音
08  boolean collision soundflag=true;
09  //当前所选关卡
10  int level;
11  //排行榜中所选关数
12  int map level index=1;
13  //当前游戏的得分
14  int curr_grade;
```

```
15 //SensorManager 对象引用
16 SensorManager mySensorManager;
17 //声音池
18 SoundPool soundPool;
19 //声音池中声音 ID 与自定义声音 ID 的 Map
20 HashMap<Integer, Integer> soundPoolMap;
21   Handler hd=new Handler(){
22         @Override
23         public void handleMessage(Message msg){
24             switch(msg.what){
25                 case 0://切换主菜单界面
26                     goToMainView();
27                 break;
28                 case 1://切换到赢的界面
29                     goToWinView();
30                     break;
31                 case 2://切换到输的界面
32                     goToFailView();
33                     break;
34                 case 3://切换到游戏的界面
35                     goToGameView();
36                     break;
37                 case 4://切换到选关界面
38                     goToMapLevelView();
39                     break;
40                 case 5://切换到设置界面
41                     goToSettingView();
42                     break;
43                 case 6://切换到排行榜界面
44                     goToRankView();
45                     break;
46             }}};
47    //初始化函数
48   @Override
49   public void onCreate(Bundle savedInstanceState){
50     super.onCreate(savedInstanceState);
51       //设置全屏显示
52       requestWindowFeature(Window.FEATURE_NO_TITLE);
53       getWindow().setFlags(
54             WindowManager.LayoutParams.FLAG_FULLSCREEN ,
55               WindowManager.LayoutParams.FLAG_FULLSCREEN
56       );
57       //强制为横屏
58       this.setRequestedOrientation(ActivityInfo
         .SCREEN_ORIENTATION_LANDSCAPE);
59       DisplayMetrics dm=new DisplayMetrics();
60       getWindowManager().getDefaultDisplay().getMetrics(dm);
61       //取得屏幕的高宽信息
62       SCREEN_HEIGHT=dm.heightPixels;
63       SCREEN_WIDTH=dm.widthPixels;
64       //获得 SensorManager 对象
65       mySensorManager = (SensorManager)getSystemService
        (SENSOR_SERVICE);
66       //初始化声音
67       initSound();
68       //初始化数据库数据
69       initDatabase();
70       //进入主菜单界面
```

```
71        goToMainView();
72    }
```

（2）程序中采用 sqlite 进行数据的存储，相应的数据库操作类 SQLiteUtil.java 如下所示。在 createOrOpenDatabase()函数中打开数据库，若数据库不存在则创建，数据库文件放在程序私有空间下面。Create Table()、delete()、insert()、update()、query()这几个函数分别用于实现数据库表的创建、数据删除、数据插入、数据更新和数据查询。

```
001 //数据库操作类
002 public class SQLiteUtil
003 {
004     static SQLiteDatabase sld;
005     //创建或打开数据库的方法
006     public static void createOrOpenDatabase()
007     {
008         try
009         {
010             sld=SQLiteDatabase.openDatabase
011             (
012                     "/data/data/com.guo.myball/mydb",
                        //当前应用程序只能在自己的包下创建数据库
013                     null,                          //CursorFactory
014                     SQLiteDatabase.OPEN READWRITE|SQLiteDatabase
                        .CREATE IF NECESSARY              //读写,若不存在则创建
015             );
016         }
017         catch(Exception e)
018         {
019             e.printStackTrace();
020         }
021     }
022     //关闭数据库
023     public static void closeDatabase()
024     {
025         try
026         {
027             sld.close();
028         }
029         catch(Exception e)
030         {
031             e.printStackTrace();
032         }
033     }
034     //建表
035     public static void createTable(String sql)
036     {
037         createOrOpenDatabase();          //打开数据库
038         try
039         {
040             sld.execSQL(sql);            //建表
041         }
042         catch(Exception e)
043         {
044             e.printStackTrace();
045         }
046         closeDatabase();                 //关闭数据库
047     }
048   //插入记录的方法
```

```
049     public static void insert(String sql)
050     {
051         createOrOpenDatabase();           //打开数据库
052         try
053         {
054             sld.execSQL(sql);
055         }
056         catch(Exception e)
057         {
058             e.printStackTrace();
059         }
060         closeDatabase();                  //关闭数据库
061     }
062     //删除记录的方法
063     public static  void delete(String sql)
064     {
065         createOrOpenDatabase();           //打开数据库
066         try
067         {
068             sld.execSQL(sql);
069         }
070         catch(Exception e)
071         {
072             e.printStackTrace();
073         }
074         closeDatabase();                  //关闭数据库
075     }
076     //修改记录的方法
077     public static void update(String sql)
078     {
079         createOrOpenDatabase();           //打开数据库
080         try
081         {
082             sld.execSQL(sql);
083         }
084         catch(Exception e)
085         {
086             e.printStackTrace();
087         }
088         closeDatabase();                  //关闭数据库
089     }
090     //查询的方法
091     public static Vector<Vector<String>> query(String sql)
092     {
093         createOrOpenDatabase();           //打开数据库
094         Vector<Vector<String>> vector=new Vector<Vector<String>>();
       //新建存放查询结果的向量
095         try
096         {
097            Cursor cur=sld.rawQuery(sql, new String[]{});
098             while(cur.moveToNext())
099             {
100                 Vector<String> v=new Vector<String>();
101                 int col=cur.getColumnCount();       //返回每一行有多少字段
102                 for( int i=0;i<col;i++)
103                 {
104                     v.add(cur.getString(i));
105                 }
106                 vector.add(v);
```

```
107            }
108            cur.close();
109        }
110        catch(Exception e)
111        {
112            e.printStackTrace();
113        }
114        closeDatabase();                    //关闭数据库
115        return vector;
116    }
117 }
```

（3）在主菜单类中还实现了各个游戏界面的切换，如下所示。代码 001～027 行将界面切换到主菜单界面，在主菜单界面中初始化 3 个按钮，并为 3 个按钮绑定监听器，单击“开始”按钮将进入选关界面，单击“排行榜”按钮将进入排行榜界面，单击“设置”按钮将进入设置界面。

代码 028～052 行实现了切换到设置界面的功能，设置完毕之后单击“确定”按钮将返回游戏菜单界面。代码 053～153 行将切换到选关界面，在该界面中主要实现 6 个 ImageButton 的按键监听函数，当单击其中一个按钮时，将初始化相应的地图并进入相应关卡。

函数 goToGameView()将实例化 GameSurfaceView 进入游戏开始界面，函数 goToRankView()将初始化 GameView，显示排行榜信息。goToWinView 和 goToFailView 分别跳转到胜利画面和失败画面，若用户当前得分超过数据库中保存的数据，将显示“刷新记录”并将分数更新到数据库中。

```
001    //进入主菜单
002    public void goToMainView(){
003     //设置当前显示界面为 main
004     setContentView(R.layout.main);
005     curr=WhichView.MAIN_MENU;
006     //初始化按钮
007     ImageButton ib_start=(ImageButton)findViewById
        (R.id.ImageButton_Start);
008     ImageButton ib_rank=(ImageButton)findViewById
       (R.id.ImageButton_Rank);
009     ImageButton ib_set=(ImageButton)findViewById
        (R.id.ImageButton_Set);
010     ib_start.setOnClickListener(            //进入到选关界面
011             new OnClickListener(){
012            @Override
013            public void onClick(View v){
014                hd.sendEmptyMessage(4);
015            }});
016     ib_rank.setOnClickListener(             //切换到排行榜界面
017             new OnClickListener(){
018            @Override
019            public void onClick(View v){
020                hd.sendEmptyMessage(6);
021            }});
022     ib_set.setOnClickListener(              //切换到设置界面
023             new OnClickListener(){
024            @Override
025            public void onClick(View v){
026                hd.sendEmptyMessage(5);
```

```
027             }});}
028     //进入设置界面
029     public void goToSettingView()
030     {
031      setContentView(R.layout.setting);
032      curr=WhichView.SETTING_VIEW;
033      //初始化选择框
034      final CheckBox cb collision=(CheckBox)findViewById
         (R.id.CheckBox collision);
035      //设置默认值
036      cb collision.setChecked(collision soundflag);
037      //OK 按钮
038      ImageButton ib_ok=(ImageButton)findViewById(R.id.ImageButton_ok);
039      ib ok.setOnClickListener
040      (
041              new OnClickListener()
042              {
043             @Override
044             public void onClick(View v)
045             {
046                 collision soundflag=cb collision.isChecked();
047                 //前往主菜单
048                 hd.sendEmptyMessage(0);
049             }
050          }
051      );
052     }
053     //进入开始游戏选关界面
054     public void goToMapLevelView()
055     {
056      //设置当前显示界面为 level_map
057      setContentView(R.layout.level_map);
058      curr=WhichView.MAPLEVEL_VIEW;
059      final ImageButton ib map[]=
060      {
061              (ImageButton)findViewById(R.id.ImageButton_map01),
062              (ImageButton)findViewById(R.id.ImageButton map02),
063              (ImageButton)findViewById(R.id.ImageButton map03),
064              (ImageButton)findViewById(R.id.ImageButton map04),
065              (ImageButton)findViewById(R.id.ImageButton map05),
066              (ImageButton)findViewById(R.id.ImageButton map06)
067      };
068          ib_map[0].setOnClickListener    //进入游戏
069          (
070              new OnClickListener()
071              {
072                 @Override
073                 public void onClick(View v)
074                 {
075                     //初始化地图数据
076                     guankaID=level=0;
077                     BallGDThread.initDiTu();
078                     hd.sendEmptyMessage(3);
079                 }
080             }
081          );
082          ib_map[1].setOnClickListener     //进入游戏
083          (
084                  new OnClickListener()
085                  {
```

```
086                 @Override
087                 public void onClick(View v)
088                 {
089                     //初始化地图数据
090                     guankaID=level=1;
091                     BallGDThread.initDiTu();
092                     hd.sendEmptyMessage(3);
093                 }
094             }
095         );
096         ib_map[2].setOnClickListener          //进入游戏
097         (
098             new OnClickListener()
099             {
100                 @Override
101                 public void onClick(View v)
102                 {
103                     //初始化地图数据
104                     guankaID=level=2;
105                     BallGDThread.initDiTu();
106                     hd.sendEmptyMessage(3);
107                 }
108             }
109         );
110         ib_map[3].setOnClickListener          //进入游戏
111         (
112             new OnClickListener()
113             {
114                 @Override
115                 public void onClick(View v)
116                 {
117                     //初始化地图数据
118                     guankaID=level=3;
119                     BallGDThread.initDiTu();
120                     hd.sendEmptyMessage(3);
121                 }
122             }
123         );
124         ib_map[4].setOnClickListener          //进入游戏
125         (
126             new OnClickListener()
127             {
128                 @Override
129                 public void onClick(View v)
130                 {
131                     //初始化地图数据
132                     guankaID=level=4;
133                     BallGDThread.initDiTu();
134                     hd.sendEmptyMessage(3);
135                 }
136             }
137         );
138         ib_map[5].setOnClickListener          //进入游戏
139         (
140             new OnClickListener()
141             {
142                 @Override
143                 public void onClick(View v)
144                 {
145                     //初始化地图数据
```

```
146                     guankaID=level=5;
147                     BallGDThread.initDiTu();
148                     hd.sendEmptyMessage(3);
149                 }
150             }
151         );
152
153     }
154     //进入游戏界面
155     public void goToGameView()
156     {
157      msv=new GameSurfaceView(this);
158      msv.requestFocus();                                    //获取焦点
159      msv.setFocusableInTouchMode(true);                     //设置为可触控
160      curr=WhichView.GAME_VIEW;
161       setContentView(msv);
162     }
163     //进入排行榜
164     public void goToRankView()
165     {
166      if(gameView==null)
167      {
168           gameView = new GameView(this);
169      }
170      setContentView(gameView);
171      curr=WhichView.RANKING_VIEW;
172     }
173     //如果闯关成功
174     public void goToWinView()
175     {
176      setContentView(R.layout.win);
177      curr=WhichView.WIN_VIEW;
178      TextView tv_score=(TextView)findViewById(R.id.TextView_score);
                                                                //当前得分
179         TextView tv_flag=(TextView)findViewById(R.id.TextView_flag);
                                                                //是否刷新纪录
180         ImageButton ib_replay=(ImageButton)findViewById
            (R.id.ImageButton_Replay);                          //“重玩”按钮
181         ImageButton ib_next=(ImageButton)findViewById
            (R.id.ImageButton_Next);                            //“下一关”按钮
182         ImageButton ib_back=(ImageButton)findViewById
            (R.id.ImageButton_Back);                            //“返回”按钮
183         tv_score.setText("本关得分为:"+curr_grade);
184         //查询本关最大的分数记录
185         String sql_maxScore="select max(grade) from rank where
            level="+(level+1);
186         System.out.println(sql_maxScore);
187      Vector<Vector<String>> vector=SQLiteUtil.query(sql_maxScore);
188      //如果当前分数大于历史记录,则刷新记录
189
190      if(vector.get(0).get(0)==null||curr_grade>Integer
         .parseInt(vector.get(0).get(0)))
191      {
192          tv_flag.setText("刷新纪录!");
193      }
194      else
195      {
196          tv_flag.setText("没有刷新纪录!");
197      }
```

```
198      insertTime(level+1,curr grade);
199      //如果当前已到达关底 则“下一关”按钮不可用
200      if(level==5)
201      {
202          ib_next.setEnabled(false);
203          ib_next.setVisibility(INVISIBLE);
204      }
205         ib_replay.setOnClickListener       //“重玩”按钮监听
206      (
207              new OnClickListener()
208              {
209             @Override
210             public void onClick(View v)
211             {
212                 BallGDThread.initDiTu();
213                 hd.sendEmptyMessage(3);
214             }
215          }
216      );
217         ib_next.setOnClickListener         //“下一关”按钮监听
218      (
219              new OnClickListener()
220              {
221             @Override
222             public void onClick(View v)
223             {
224                 if(level<5)
225                 {
226                     level++;
227                 }
228                 guankaID=level;            //“进入”下一关
229                 BallGDThread.initDiTu();
230                 hd.sendEmptyMessage(3);
231             }
232          }
233      );
234         ib back.setOnClickListener         //“返回”按钮监听，返回到选关界面
235      (
236              new OnClickListener()
237              {
238             @Override
239             public void onClick(View v)
240             {
241                 hd.sendEmptyMessage(4);
242             }
243          }
244      );
245     }
246     //如果闯关失败
247     public void goToFailView()
248     {
249      setContentView(R.layout.fail);
250      curr=WhichView.FAIL VIEW;
251         ImageButton ib replay=(ImageButton)findViewById
            (R.id.Fail ImageButton Replay);
252         ImageButton ib_back=(ImageButton)findViewById
            (R.id.Fail ImageButton Back);
253         ib_replay.setOnClickListener           //“重玩”按钮监听
254      (
255              new OnClickListener()
```

```
256              {
257             @Override
258             public void onClick(View v)
259             {
260                 BallGDThread.initDiTu();
261                 hd.sendEmptyMessage(3);
262             }
263         }
264     );
265       ib_back.setOnClickListener        //“返回”按钮监听返回到选关界面
266     (
267              new OnClickListener()
268              {
269             @Override
270             public void onClick(View v)
271             {
272                 hd.sendEmptyMessage(4);
273             }
274         }
275     );
276    }
```

（4）由于本游戏是采用 GSensor 对小球进行控制，因此在这里创建了传感器类 mySensorListener，在该类中监听传感器变化产生的数据，并提取出 X、Y 轴上的加速度。然后在 onResume()函数中将传感器监听器注册上，在 onPause()函数中将传感器监听器注销。

```
01  private SensorListener mySensorListener = new SensorListener(){
02      @Override
03      public void onAccuracyChanged(int sensor, int accuracy)
04      {
05      }
06      @Override
07      public void onSensorChanged(int sensor, float[] values)
08      {
09          if(sensor == SensorManager.SENSOR_ORIENTATION)
10          {//判断是否为加速度传感器变化产生的数据
11              int directionDotXY[]=RotateUtil.getDirectionDot
12              (
13                      new double[]{values[0],values[1],values[2]}
14              );
15
16              ballGX=-directionDotXY[0]*3.2f;       //得到 X 和 Y 方向上的加速度
17              ballGZ=directionDotXY[1]*3.2f;
18          }
19      }
20  };
21    @Override
22  protected void onResume()                         //重写 onResume 方法
23    {
24      super.onResume();
25      mySensorManager.registerListener
26      (                                             //注册监听器
27              mySensorListener,                     //监听器对象
28              SensorManager.SENSOR_ORIENTATION,     //传感器类型
29              SensorManager.SENSOR_DELAY_UI         //传感器事件传递的频度
30      );
31  }
```

```
32  @Override
33  protected void onPause()                              //重写 onPause 方法
34  {
35      super.onPause();
36      mySensorManager.unregisterListener(mySensorListener);
                                                          //取消注册监听器
37  }
```

（5）游戏碰撞声音初始化在 initSound 中实现，如下所示，将 raw 文件下的 dong.mp3 添加到声音池中，当需要播放该声音时，直接调用 playSound 就可以。

```
01  //初始化声音
02  public void initSound()
03    {
04          //声音池
05          soundPool = new SoundPool(4, AudioManager.STREAM_MUSIC, 100);
06          soundPoolMap = new HashMap<Integer, Integer>();
07          //碰撞音乐
08          soundPoolMap.put(1, soundPool.load(this, R.raw.dong, 1));
09    }
10    //播放声音
11    public void playSound(int sound, int loop)
12    {
13     if(collision_soundflag)
14     {
15         AudioManager mgr = (AudioManager)this.getSystemService
          (Context.AUDIO_SERVICE);
16         //取得当前音量
17          float streamVolumeCurrent = mgr.getStreamVolume(AudioManager.
          STREAM_MUSIC);
18          //取得最大音量
19          float streamVolumeMax = mgr.getStreamMaxVolume(AudioManager.
          STREAM_MUSIC);
20          //设置音量
21          float volume = streamVolumeCurrent / streamVolumeMax;
22          //播放音乐
23          soundPool.play(soundPoolMap.get(sound), volume, volume, 1, loop,
         1f);
24     }
25  }
```

（6）游戏排行榜界面的绘制我们可以采用 layout 下的 xml 文件，也可以通过 SurfaceView 进行绘制，前者比较方便也易于操作，但是为了多演示一些绘制的功能，在此处我们选择后面一种方式。在类的构造函数中将调用函数 initBitmap()初始化图片资源，在 surfaceCreated()函数中调用函数 onDraw 绘制画面。在 onDraw()函数中将根据当前设置的关数到数据库中查询对应关数的分数数据，并绘制到指定的位置。在 onTouch()函数中实现对加减关数按钮的监听，单击减号则显示上一关的分数，单击加号则显示下一关的分数。

```
001 //分数显示界面
002 public class GameView extends SurfaceView implements SurfaceHolder.
    Callback{
003     MapMasetActivity activity;      //Activity 引用
004     Canvas c;
005     SurfaceHolder holder;
006     int scoreWidth = 10;
007     int guanshuX;                  //关数文字 X 坐标
```

```
008     int guanshuY;                           //关数文字 Y 坐标
009     int guanshu=1;
010     Bitmap iback;                           //背景图
011     Bitmap[] iscore=new Bitmap[10];         //得分图
012     Bitmap JianHaotupian;                   //减号图
013     Bitmap JiaHaotupian;                    //加号图
014     Bitmap[] guanShu=new Bitmap[10];        //关数文字
015     Bitmap time_wz;                         //时间文字图
016     Bitmap gread_wz;                        //成绩文字图
017     Bitmap hengXian;                        //横线
018     public GameView(MapMasetActivity activity) {
019         super(activity);
020         getHolder().addCallback(this);      //注册回调接口
021         this.activity = activity;
022         initBitmap();
023     }
024     //将图片加载
025     public void initBitmap(){
026         iback = BitmapFactory.decodeResource(getResources(),
            R.drawable.main);
027         iscore[0] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d0);                     //数字图
028         iscore[1] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d1);
029         iscore[2] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d2);
030         iscore[3] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d3);
031         iscore[4] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d4);
032         iscore[5] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d5);
033         iscore[6] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d6);
034         iscore[7] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d7);
035         iscore[8] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d8);
036         iscore[9] = BitmapFactory.decodeResource(getResources(),
            R.drawable.d9);
037
038         guanShu[0] = BitmapFactory.decodeResource(getResources(),
            R.drawable.guanka);                 //关数文字
039         guanShu[1] = BitmapFactory.decodeResource(getResources(),
            R.drawable.guanka1);
040         guanShu[2] = BitmapFactory.decodeResource(getResources(),
            R.drawable.guanka2);
041         guanShu[3] = BitmapFactory.decodeResource(getResources(),
            R.drawable.guanka3);
042         guanShu[4] = BitmapFactory.decodeResource(getResources(),
            R.drawable.guanka4);
043         guanShu[5] = BitmapFactory.decodeResource(getResources(),
            R.drawable.guanka5);
044         JiaHaotupian = BitmapFactory.decodeResource(getResources(),
            R.drawable.right);
045         JianHaotupian = BitmapFactory.decodeResource(getResources(),
            R.drawable.left);
046         //成绩文字
```

```
047        gread wz = BitmapFactory.decodeResource(getResources(),
           R.drawable.grade);
048        //时间文字
049        time wz= BitmapFactory.decodeResource(getResources(),
           R.drawable.time);
050        //横线
051        hengXian=BitmapFactory.decodeResource(getResources(),
           R.drawable.hengxian);
052    }
053    @Override
054    protected void onDraw(Canvas canvas)
055    {
056        super.onDraw(canvas);
057        canvas.drawColor(Color.argb(255, 0, 0, 0));
058        //画背景
059        canvas.drawBitmap(iback,30,0, null);
060        //绘制减号和加号图片
061        canvas.drawBitmap(JianHaotupian,SCREEN_WIDTH/6,
           SCREEN_HEIGHT/6+40, null);
062        //绘制关卡文字
063        canvas.drawBitmap(guanShu[guanshu-1],SCREEN WIDTH
          /2-60,SCREEN HEIGHT/6+40, null);
064        //绘制右边加号
065        canvas.drawBitmap(JiaHaotupian,SCREEN_WIDTH
          /2+80,SCREEN_HEIGHT/6+40, null);
066        //绘制成绩文字 gread_wz
067        canvas.drawBitmap(gread wz,SCREEN WIDTH/6,SCREEN HEIGHT
          /6+63, null);
068        //绘制游戏时间文字
069        canvas.drawBitmap(time_wz,SCREEN_WIDTH/2+80,SCREEN_HEIGHT
          /6+63, null);
070        //从数据库中取出相应的数据
071        String sql_select="select grade,time from rank where
           level="+guanshu+" order by grade desc limit 0,5;";
072        Vector<Vector<String>> vector=SQLiteUtil.query(sql select);
073        //循环绘制排行榜的分数和对应时间
074        for(int i=0;i<vector.size();i++)
075        {
076            //成绩，日期
077            drawScoreStr(canvas,vector.get(i).get(0).toString(),
               SCREEN_WIDTH/6,SCREEN_HEIGHT/6+40+60+i*30);
078            drawRiQi(canvas,vector.get(i).get(1).toString(),
               SCREEN_WIDTH/2+65,SCREEN_HEIGHT/6+40+60+i*30);
079        }
080    }
081    //绘制字符串方法
082    public void drawScoreStr(Canvas canvas,String s,int width,int
       height)
083    {
084        //绘制得分
085        String scoreStr=s;
086        //循环绘制得分
087        for(int i=0;i<scoreStr.length();i++){
088            int tempScore=scoreStr.charAt(i)-'0';
089            canvas.drawBitmap(iscore[tempScore], width+i*scoreWidth,
               height, null);
090            }
091    }
092    //画年月
```

```
093     public void drawRiQi(Canvas canvas,String s,int width,int height)
094     {
095         //切割得到年月日
096         String ss[]=s.split("-");
097         //画年数数字
098         drawScoreStr(canvas,ss[0],width,height);
099         //画横线
100         canvas.drawBitmap(hengXian,width+scoreWidth*4,height, null);
101         //画月数数字
102         drawScoreStr(canvas,ss[1],width+scoreWidth*5,height);
103         //画横线
104         canvas.drawBitmap(hengXian,width+scoreWidth*7,height, null);
105         //画日数数字
106         drawScoreStr(canvas,ss[2],width+scoreWidth*8,height);
107     }
108     @Override
109     public boolean onTouchEvent(MotionEvent event){
110         int x = (int) event.getX();
111         int y = (int) event.getY();
112         if(x>SCREEN WIDTH/6&&x<SCREEN WIDTH/6+60&&
113                 y>SCREEN HEIGHT/6+40&&y<SCREEN HEIGHT/6+40+40)
114         {
115             if(guanshu>1)
116             {
117                 guanshu--;
118                 c = null;
119                 try {
120                     //锁定整个画布，在内存要求比较高的情况下，建议参数不要为null
121                     c = holder.lockCanvas(null);
122                     synchronized (holder) {
123                         onDraw(c);//绘制
124                     }
125                 } finally {
126                     if (c != null) {
127                         //并释放锁
128                         holder.unlockCanvasAndPost(c);
129                     }
130                 }
131             }
132         }
133         if(x>SCREEN WIDTH/2+80&&x<SCREEN WIDTH/2+140
134                 &&y>SCREEN HEIGHT/6+40&&y<SCREEN HEIGHT/6+80){
135             if(guanshu<MAPP.length)
136             {
137                 guanshu++;
138                 c = null;
139                 try {
140                     //锁定整个画布，在内存要求比较高的情况下，建议参数不要为null
141                     c = holder.lockCanvas(null);
142                     synchronized (holder) {
143                         onDraw(c);          //绘制
144                     }
145                 } finally {
146                     if (c != null) {
147                         //并释放锁
148                         holder.unlockCanvasAndPost(c);
149                     }
150                 }
151             }
```

```
152         }
153         return super.onTouchEvent(event);
154     }
155     public void surfaceChanged(SurfaceHolder holder, int format, int
        width, int height) {}
156     public void surfaceCreated(SurfaceHolder holder){//创建时启动相应进程
157         this.holder=holder;
158             c = null;
159             try {
160                 //锁定整个画布，在内存要求比较高的情况下，建议参数不要为 null
161                 c = holder.lockCanvas(null);
162                 synchronized (holder) {
163                     onDraw(c);          //绘制
164                 }
165             } finally {
166                 if (c != null) {
167                     //并释放锁
168                     holder.unlockCanvasAndPost(c);
169                 }
170             }
171     }
172     public void surfaceDestroyed(SurfaceHolder holder) {
        //摧毁时释放相应进程
173     }
174 }
```

22.4　游 戏 进 行

（1）游戏画面绘制类如下所示，在类的开始声明所有需要用到的变量，包括关卡信息、摄像机信息、球的信息、墙壁、圆洞等。接着在构造函数中初始化地图信息，包括地图组数、圆洞数组、小球初始位置、小球目标位置等，并设置摄像机的位置，实例化场景渲染器。

```
01  //游戏进行界面
02  class GameSurfaceView extends GLSurfaceView
03  {
04      MapMasetActivity father;                        //声明 Activity
05      public static int guankaID;                     //关卡 ID
06      public static int[][]MAP;                       //对应关卡的地图数组
07      public static int[][] MAP_OBJECT;               //对应关卡的洞数组
08      public static int STIME;                        //每一关对应的时间限制
09
10      public static float yAngle=0f;                  //方位角
11      public static float xAngle=90f;                 //仰角
12      public static float cx;                         //摄像机 x 坐标
13      public static float cy;                         //摄像机 y 坐标
14      public static float cz;                         //摄像机 z 坐标
15      public static float tx=0;                       //观察目标点 x 坐标
16      public static float ty=0;                       //观察目标点 y 坐标
17      public static float tz=0f;                      //观察目标点 z 坐标
18      public static float upX=0;
19      public static float upY=1;
20      public static float upZ=0;                      //up 轴
```

```
21
22      public static float ballX;                  //球的各个坐标
23      public static float ballY;
24      public static float ballZ;
25      public static float ballGX=0f;              //x 方向上的加速度
26      public static float ballGZ=0f;              //y 方向上的加速度
27
28      public static int ballCsX;                  //初始格子
29      public static int ballCsZ;
30      public static int ballMbX;                  //目标格子
31      public static int ballMbZ;
32
33      public static float ballVX=0;               //XZ 方向上的速度
34      public static float ballVZ=0;
35
36      private SceneRenderer mRenderer;            //场景渲染器
37
38      public static int floorId;                  //地板纹理 ID
39      public static int wallId;                   //墙纹理
40      public static int yuankonId;                //圆孔纹理 ID
41      public static int ballId;                   //球纹理 ID
42      public static int ballYZId;                 //球的影子纹理 ID
43      public static int numberId;                 //数字 ID
44      public static int time_DH_Id;               //顿号 ID
45      public static int mbyuankonId;
46
47      public RectWall yuankon;                    //圆孔矩形
48      public Floor floor;                         //地板
49      public static  Wall wall;                   //墙
50      public BallTextureByVertex ball;            //球
51      public RectWall ballYZ;                     //球的影子矩形
52      public Number number;                       //数字
53      public TextureRect time_DH;                 //顿号，用于时间
54
55      BallGDThread ballgdT;                       //球运动线程
56
57      public GameSurfaceView(Context context)
58      {
59          super(context);
60          this.father=(MapMasetActivity)context;
61          ballCsX=CAMERA_COL_ROW[guankaID][0];        //初始行列
62          ballCsZ=CAMERA_COL_ROW[guankaID][1];
63
64          ballMbX=CAMERA_COL_ROW[guankaID][2];        //目标行列
65          ballMbZ=CAMERA_COL_ROW[guankaID][3];
66
67          MAP=MAPP[guankaID];                         //地图数组
68          MAP_OBJECT=MAP_OBJECTT[guankaID];           //洞数组
69          STIME=GD_TIME[guankaID];                    //限制时间
70
71          ballX=ballCsX*UNIT_SIZE-MAP[0].length*UNIT_SIZE/2;
                                                        //初始化球位置
72          ballZ=ballCsZ*UNIT_SIZE-MAP.length*UNIT_SIZE/2;
73          ballY=ballR;
74
75          tx=0;                                       //摄像机目标位置
76          ty=0;
```

```
77          tz=0;
78          ballgdT=new BallGDThread(this);
79          //设置摄像机的位置
80          cx=(float)(tx+Math.cos(Math.toRadians(xAngle))*Math.sin
            (Math.toRadians(yAngle))*DISTANCE);        //摄像机 x 坐标
81          cz=(float)(tz+Math.cos(Math.toRadians(xAngle))*Math.
            cos(Math.toRadians(yAngle))*DISTANCE);   //摄像机 z 坐标
82          cy=(float)(ty+Math.sin(Math.toRadians(xAngle))*DISTANCE);
              //摄像机 y 坐标
83          mRenderer = new SceneRenderer();           //创建场景渲染器
84          setRenderer(mRenderer);                    //设置渲染器
85          setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
                                                       //设置渲染模式为主动渲染
86
87      }
```

（2）场景渲染器实现如下所示，这个类是绘制游戏画面的核心类，程序首先执行 onSurfaceCreated 回调函数，如代码 097～142 行所示。在函数 onSurfaceCreated()中首先开启 opengl 绘图相关的模式，做一些绘图前的准备，接着初始化地面、墙、圆孔、球、数字等的纹理。之后初始化地板、墙、圆孔、小球、数字等类，并启动小球运动线程，初始化灯光，这样绘制的准备工作都已完成。

执行完 onSurfaceCreated()函数之后将执行 onDrawFrame()函数绘制游戏动画帧，如代码 005～061 行所示。绘制时先设置摄像机的位置和一些其他相关的属性，之后调用各个类的绘图函数将图片绘制出来。函数 drawBall()和 drawYuanKong()分别用来绘制小球和圆孔。

```
001 //场景渲染器
002 private class SceneRenderer implements GLSurfaceView.Renderer
003 {
004     //绘制帧
005         public void onDrawFrame(GL10 gl)
006         {
007          //采用平滑着色
008          gl.glShadeModel(GL10.GL_SMOOTH);
009          //清除颜色缓存
010          gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_
             BUFFER_BIT);
011          //设置当前矩阵为模式矩阵
012              gl.glMatrixMode(GL10.GL_MODELVIEW);
013              //设置当前矩阵为单位矩阵
014              gl.glLoadIdentity();
015              //设置 camera 位置
016              GLU.gluLookAt
017              (gl, cx,cy,cz, tx,ty, tz,0,1, 0);
                                                     //设置摄像机
018              gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
                                                     //启用顶点数组
019              gl.glEnable(GL10.GL_TEXTURE_2D);
                                                     //启用纹理
020              gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
021
022              gl.glEnable(GL10.GL_LIGHTING);
                                                     //允许光照
023              gl.glEnable(GL10.GL_LIGHT0);        //开 0 号灯
```

```
024                                                    //允许使用法向量数组
025         gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);
026
027         floor.drawSelf(gl, floorId);                      //绘制地板
028
029         gl.glPushMatrix();                                //保护矩阵
030         gl.glTranslatef(-MAP[0].length/2*UNIT_SIZE, 0,
           (-MAP.length/2)*UNIT_SIZE);
031         wall.drawSelf(gl, wallId);                        //绘制墙
032         gl.glPopMatrix();                                 //恢复矩阵
033
034         gl.glDisable(GL10.GL_LIGHTING);                   //关闭光照
035         gl.glDisableClientState(GL10.GL_NORMAL_ARRAY)
                                                              //关闭法向量数组
036
037         gl.glEnable(GL10.GL_BLEND);
                                                              //开启混合
038
039         gl.glBlendFunc(GL10.GL_SRC_ALPHA ,GL10.GL_ONE_MINUS
            SRC_ALPHA);                                       //设置混合因子
040         gl.glPushMatrix();                                //保护当前矩阵
041         gl.glTranslatef(ballMbX*UNIT_SIZE-MAP[0].
           length*UNIT_SIZE/2,
042              0.015f,
043              ballMbZ*UNIT_SIZE- MAP.length*UNIT_SIZE/2);
044         gl.glRotatef(-90, 1, 0, 0);
045         yuankon.drawSelf(gl, mbyuankonId);         //绘制目标圆孔
046         gl.glPopMatrix();
047         drawYuanKong(gl);                          //绘制圆孔
048         gl.glPushMatrix();
049         gl.glTranslatef(ballX+ballR-0.2f, 0.01f, ballZ-ballR+0.2f);
050         gl.glRotatef(-90, 1, 0, 0);
051         gl.glRotatef(45, 0, 0, 1);
052         ballYZ.drawSelf(gl, ballYZId);         //绘制影子
053         gl.glPopMatrix();
054         gl.glDisable(GL10.GL_BLEND);           //关闭混合
055         drawBall(gl);                          //绘制球
056         drawNumber(gl);                        //绘制当前剩余时间数字
057
058         gl.glDisableClientState(GL10.GL_VERTEX_ARRAY); //关闭顶点数组
059         gl.glDisable(GL10.GL_TEXTURE_2D);                //关闭纹理
060         gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
061      }
062
063     public  void drawNumber(GL10 gl)                  //绘制剩余时间方法
064     {
065      gl.glMatrixMode(GL10.GL_MODELVIEW);              //模式矩阵
066      gl.glLoadIdentity();          //设置当前矩阵为单位矩阵
067
068      gl.glEnable(GL10.GL_BLEND);                      //开启混合
069      gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
         //设置混合因子
070      //绘制数字仪表盘高度
071      gl.glPushMatrix();
072      gl.glTranslatef(2.0f,1.6f,-6);        //设置仪表板的位置不能再调节
073      number.y=0;                           //数字的 Y 坐标
```

```
074         number.NumberStr=Math.abs(GD TIME[guankaID]-STIME)/60+"";
                                                    //剩下的分钟数
075         number.drawSelf(gl,0,numberId);         //绘制分钟
076         gl.glTranslatef(ICON_WIDTH*0.7f,0f,0);
077         time_DH.drawSelf(gl, time_DH_Id);               //画顿号
078         gl.glTranslatef(ICON_WIDTH*0.7f,0f,0);
079         number.NumberStr=Math.abs(GD TIME[guankaID]-STIME)%60+"";
080         number.drawSelf(gl,1,numberId);                 //画秒数
081         gl.glPopMatrix();                               //恢复矩阵
082         gl.glDisableClientState(GL10.GL BLEND);         //关闭混合
083        }
084        public void onSurfaceChanged(GL10 gl, int width, int height)
085        {
086            //设置视窗大小及位置
087            gl.glViewport(0, 0, width, height);
088            //设置当前矩阵为投影矩阵
089            gl.glMatrixMode(GL10.GL PROJECTION);
090            //设置当前矩阵为单位矩阵
091            gl.glLoadIdentity();
092            //计算透视投影的比例
093            float ratio = (float) width / height;
094            //调用此方法计算产生透视投影矩阵
095            gl.glFrustumf(-ratio, ratio, -1, 1, 3, 1000);
096        }
097        public void onSurfaceCreated(GL10 gl, EGLConfig config)
098        {
099            //关闭抗抖动
100            gl.glDisable(GL10.GL DITHER);
101            //设置特定 Hint 项目的模式，这里设置为使用快速模式
102            gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,GL10.
               GL_FASTEST);
103            //设置为打开背面剪裁
104            gl.glEnable(GL10.GL CULL FACE);
105            //设置着色模型为平滑着色
106            gl.glShadeModel(GL10.GL_SMOOTH);
107            //开启混合
108            //设置屏幕背景色为黑色 RGBA
109            gl.glClearColor(0,0,0,0);
110            //启用深度测试
111            gl.glEnable(GL10.GL_DEPTH_TEST);
112            floorId=initTexture(gl,R.drawable.floor);        //地面 ID
113            wallId=initTexture(gl,R.drawable.wall);          //墙 ID
114
115            yuankonId=initTexture2(gl,R.drawable.yuankon);   //圆孔 ID
116            ballId=initTexture2(gl,R.drawable.ball);         //球 ID
117            ballYZId=initTexture2(gl,R.drawable.ballyingzi); //球的影子
ID
118            numberId=initTexture2(gl,R.drawable.number);     //数字 ID
119            time_DH_Id=initTexture2(gl,R.drawable.dunhao);   //顿号纹理
120            mbyuankonId=initTexture2(gl,R.drawable.mbyuankon);
                                                           //目标圆孔 ID
121
122
123            floor=new Floor(MAP[0].length,MAP.length);
                                                                  //地板
124            wall=new Wall();                                   //墙
```

```
125             yuankon=new RectWall(2f*ballR,2f*ballR);                //圆孔
126             ball=new BallTextureByVertex(ballR,15);                 //球
127             ballYZ=new RectWall(3.6f*ballR,2.6f*ballR               //影子
128             number=new Number(GameSurfaceView.this);                //数字对象
129             time_DH=new TextureRect(ICON_WIDTH*0.5f/2,              //数字
130               ICON_HEIGHT*0.5f/2,
131                   new float[]
132                   {
133                    0,0, 0,1, 1,0,
134                    0,1, 1,1,  1,0
135                   });                                            //顿号
136             ballgdT.start();
137 
138             initLight(gl);                                           //初始化灯光
139             float[] positionParamsGreen={-4,4,4,0};//最后的 0 表示是定向光
140             gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, position
                ParamsGreen,0);
141 
142         }
143         public void drawBall(GL10 gl)                            //画重力球
144         {
145          gl.glPushMatrix();
146          gl.glTranslatef(ballX, ballY, ballZ);                   //移动相应的位置
147          ball.drawSelf(gl, ballId);   //绘制
148          gl.glPopMatrix();
149         }
150         public void drawYuanKong(GL10 gl)                        //绘制圆孔
151         {
152          gl.glPushMatrix();
153          gl.glTranslatef(-MAP[0].length*UNIT_SIZE/2, 0.01f,-
             MAP.length*UNIT_SIZE/2);
154          for(int i=0;i<MAP_OBJECT.length;i++)
155          {
156              for(int j=0;j<MAP_OBJECT[0].length;j++)
157              {
158                  if(MAP_OBJECT[i][j]==1)
159                  {
160                      if(i==ballMbX&&j==ballMbZ)     //如果不是目标洞则绘制
161                      {
162                         continue;
163                      }
164                      gl.glPushMatrix();
165                      gl.glTranslatef((j)*UNIT_SIZE, 0.001f,
                        (i)*UNIT_SIZE);
166                      gl.glRotatef(-90, 1, 0, 0);
167                      yuankon.drawSelf(gl, yuankonId);            //绘制
168                      gl.glPopMatrix();
169                  }
170              }
171          }
172          gl.glPopMatrix();
173         }
174         private void initLight(GL10 gl)
175     {
```

```
176             //白色灯光
177             gl.glEnable(GL10.GL_LIGHT0);                    //打开 0 号灯
178             //环境光设置
179             float[] ambientParams={1f,1f,1f,1.0f};          //光参数 RGBA
180             gl.glLightfv(GL10.GL LIGHT0, GL10.GL AMBIENT,
                ambientParams,0);
181             //散射光设置
182             float[] diffuseParams={1f,1f,1f,1.0f};          //光参数 RGBA
183             gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_DIFFUSE,
                diffuseParams,0);
184             //反射光设置
185             float[] specularParams={1f,1f,1f,1.0f};         //光参数 RGBA
186             gl.glLightfv(GL10.GL LIGHT0, GL10.GL SPECULAR,
                specularParams,0);
187       }
188   }
```

（3）在绘制图片时用到了初始化纹理的函数 initTexture2 和 initTexture，这两个函数的功能非常类似，实现代码页基本相同，唯一不同的就是前一种的纹理采用拉伸的方式，后一种的纹理采用平铺的方式。initTexture 主要用于需要重复使用的纹理，如地板、墙壁，而 initTexture 则用于需要拉伸显示的纹理，如小球、圆孔等。

```
01  //初始化纹理
02  public int initTexture2(GL10 gl,int drawableId)//textureId
03  {
04      //生成纹理 ID
05      int[] textures = new int[1];
06      gl.glGenTextures(1, textures, 0);
07      int currTextureId=textures[0];
08      gl.glBindTexture(GL10.GL TEXTURE 2D, currTextureId);
09      //在 MIN_FILTER MAG_FILTER 中使用 MIPMAP 纹理
10      gl.glTexParameterf(GL10.GL TEXTURE 2D,
        GL10.GL TEXTURE MIN FILTER,GL10.GL NEAREST);
11        gl.glTexParameterf(GL10.GL_TEXTURE_2D,GL10.GL_TEXTURE
          MAG FILTER,GL10.GL LINEAR);
12        //纹理拉伸
13        gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S
         ,GL10.GL_CLAMP_TO_EDGE);
14        gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
          GL10.GL_CLAMP_TO_EDGE);
15        //获得图片资源
16        InputStream is = this.getResources().openRawResource(drawableId);
17        Bitmap bitmapTmp;
18        try
19        {
20          bitmapTmp = BitmapFactory.decodeStream(is);
21        }
22        finally
23        {
24            try
25            {
26                is.close();
27            }
28            catch(IOException e)
29            {
30                e.printStackTrace();
31            }
32        }
```

```
33          //生成纹理
34          GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmapTmp, 0);
35          //回收资源
36          bitmapTmp.recycle();
37          return currTextureId;
38   }
39
40   //初始化纹理
41   public int initTexture(GL10 gl,int drawableId)//textureId
42   {
43       //生成纹理 ID
44       int[] textures = new int[1];
45       gl.glGenTextures(1, textures, 0);
46       int currTextureId=textures[0];
47       //绑定纹理
48       gl.glBindTexture(GL10.GL_TEXTURE_2D, currTextureId);
49
50       //在 MIN_FILTER MAG_FILTER 中使用 MIPMAP 纹理
51       gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN
         _FILTER, GL10.GL_LINEAR_MIPMAP_NEAREST);
52       gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_
         MAG_FILTER, GL10.GL_LINEAR_MIPMAP_LINEAR);
53       //生成 Mipmap 纹理
54       ((GL11)gl).glTexParameterf(GL10.GL_TEXTURE_
          2D,GL11.GL_GENERATE_MIPMAP,GL10.GL_TRUE);
55          //纹理平铺
56          gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
            GL10.GL_REPEAT);
57          gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
            GL10.GL_REPEAT);
58          //取得图片资源
59          InputStream is = this.getResources().openRawResource(drawableId);
60          Bitmap bitmapTmp;
61          try
62          {
63            bitmapTmp = BitmapFactory.decodeStream(is);
64          }
65          finally
66          {
67              try
68              {
69                  is.close();
70              }
71              catch(IOException e)
72              {
73                  e.printStackTrace();
74              }
75          }
76          //生成纹理
77          GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmapTmp, 0);
78          //回收资源
79          bitmapTmp.recycle();
80          return currTextureId;
81   }
```

（4）小球运动轨迹及碰撞情况的判断均在类 BallGDThread 中实现，类 BallGDThread 继承于 Thread，线程的执行函数 run 如下所示。变量 flag 为游戏结束的标志，flag 为 false 表示游戏结束。函数的开始先从 GameSurfaceView 中复制 XZ 轴的加速度，并根据加速度

和小球位置等参数调用 PDPZ 函数判断小球的碰撞情况。接着计算出小球经过 t 时间后的坐标，并调用函数 PDJD 判断小球是否进洞，如果进洞了则进一步判断是否进入了目标洞内。若进入目标洞内，则进入胜利界面，否则游戏结束，进入失败界面。

若小球未掉进洞内，则进一步计算小球的速度，更新当前倒计时时间，若时间耗尽则游戏结束，进入失败界面，否则休眠 50 毫秒进入下一次循环。

```
01      @Override
02      public void run()
03      {
04          while(flag)
05          {
06              ballGX=GameSurfaceView.ballGX;//复制加速度
07              ballGZ=GameSurfaceView.ballGZ;
08              try{
09              //判断碰撞情况方法
10              PDPZ(ballX,ballZ,ballVX*t+ballGX*t*t/2,ballVZ*t+
                ballGZ*t*t/2);
11              }catch (Exception tt) {
12                  tt.printStackTrace();
13              }
14              ballVX+=ballGX*t;
15              ballVZ+=ballGZ*t;              //最终速度
16
17              ballX=ballX+ballVX*t+ballGX*t*t/2;//VT+1/2A*T*T
18              ballZ=ballZ+ballVZ*t+ballGZ*t*t/2;                //最终位置
19              gameSurface.ball.mAngleX+=(float)Math.
                toDegrees(((ballVZ*t+ballGZ*t*t/2))/ballR);
20              gameSurface.ball.mAngleZ-=(float)Math.toDegrees
               ((ballVX*t+ballGX*t*t/2)/ballR);//旋转的角度
21              if(Math.abs((ballVZ*t+ballGZ*t*t/2))<0.005f)
                              //如果当前前进值小于调整值，则相应的转动方向角归零
22              {
23                  gameSurface.ball.mAngleX=0;
24              }
25              if(Math.abs(ballVX*t+ballGX*t*t/2)<0.005f)
26              {
27                  gameSurface.ball.mAngleZ=0;
28              }
29              //判断进洞函数及相应的操作
30              PDJD();
31              if(!flagSY)                                 //如果掉进洞里了
32              {
33                  flagSY=true;//重新初始化小球的位置
34                  if(ballXx==ballMbX&&ballZz==ballMbZ)      //如果是赢了
35                  {
36                      try
37                      {
38                          Thread.sleep(1000);
39                      }
40                      catch (InterruptedException e)
41                      {
42                          e.printStackTrace();
43                      }
44                      flag=false;
45                      gameSurface.father.curr grade=GD TIME[guankaID]
                        -STIME;
```

```
46                      gameSurface.father.hd.sendEmptyMessage(1);
                                                       //进入赢的界面
47                  }
48                  else//否则是进洞了
49                  {
50                      try
51                      {
52                          Thread.sleep(1000);                    //停顿 1 秒
53                      } catch (InterruptedException e)
54                      {
55                          e.printStackTrace();
56                      }
57                  ballX=ballCsX*UNIT_SIZE-MAP[0].length*UNIT_SIZE/2;
58                  ballZ=ballCsZ*UNIT_SIZE-MAP.length*UNIT_SIZE/2;
59                  ballY=ballR;
60                  }
61                  ballVX=0;
62                  ballVZ=0;                    //最终速度都减为零
63              }
64              ballVX*=V_TENUATION;
65              ballVZ*=V_TENUATION;            //衰减
66              if(Math.abs(ballVX)<0.04)       //当速度小于某个调整值时
67              {
68                  ballVX=0;                           //速度归零
69                  gameSurface.ball.mAngleZ=0;         //将绕轴旋转的值置为零
70              }
71              if(Math.abs(ballVZ)<0.04)
72              {
73                  ballVZ=0;
74                  gameSurface.ball.mAngleX=0;
75              }
76              //每局的总时间
77              ZJS_Time+=50;
78              //走过的时间秒数
79              STIME=(ZJS_Time/1000);
80              //如果时间减到零，说明没有通过
81              if(GD_TIME[guankaID]-STIME<=0)
82              {
83                  flag=false;
84                  gameSurface.father.hd.sendEmptyMessage(2);
85              }
86              try //休息 50 毫秒，进入下一次循环
87              {
88                  Thread.sleep(50);
89              }
90              catch (InterruptedException e)
91              {
92                  e.printStackTrace();
93              }
94          }
95      }
```

（5）判断碰撞的函数如下所示，根据 BZ 和 BX 的值分 4 种情况进行判断，每种情况的判断方式类似。碰撞有两种方式，一种是垂直于 X 轴和 Z 轴方向的碰撞，这种碰撞只会导致其中一个分速度发生改变；另一种碰撞是球与墙角的碰撞，这种碰撞会同时改变球的 X 轴速度和 Y 轴速度。

```
001 //判断碰撞
002 public Boolean PDPZ(float ballX,float ballZ,float BX,float BZ)
003 {
004     Boolean flag=false;
005     ballX=MAP[0].length*UNIT_SIZE/2+ballX;//将地图移到 X、Z 都大于零的象限
006     ballZ=MAP.length*UNIT_SIZE/2+ballZ;
007     if(BZ>0)                                    //如果向 Z 轴正方向运动
008     {
009         for(int i=(int)((ballZ+ballR)/UNIT_SIZE);i<=(int)
            ((ballZ+ballR+BZ)/UNIT SIZE);i++)
010             //循环，假如它一下穿过几个格子，那么从第一个格子开始判断
011         {
012             if(MAP[i][(int)(ballX/UNIT_SIZE)]==BKTG&&MAP[i-1]
                [(int)(ballX/UNIT_SIZE)]==KTG){//判断是否碰墙壁了
013                 ballVZ=-ballVZ*VZ_TENUATION;      //将速度置反，并调整
014                 if((GameSurfaceView.ballZ+ballVZ*t+ballGZ*t*t/2)
                    >=(i*UNIT_SIZE-ballR-MAP.length*UNIT_SIZE/2))
                    //如果速度调反后还是会穿墙，那么将加速度归零，并将球画在和墙壁紧
                    挨着的地方
015                 {
016                     GameSurfaceView.ballZ=(i*UNIT_SIZE-ballR-MAP.
                        length*UNIT_SIZE/2);
017                     ballVZ=0;
018                     ballGZ=0;
019                 }
020                 else{
021                 gameSurface.father.playSound(1, 0);           //播放移动
                    声音
022                 }
023                 flag=true;//标志位置为 true
024             }
025             else if(BX<=0&&((int)((ballX-ballR)/UNIT SIZE)
            >=0)&&(MAP[i][(int)((ballX-ballR)/UNIT SIZE)]==BKTG)
026                 &&MAP[i-1][(int)((ballX-ballR)/UNIT SIZE)]==KTG)
                                    //如果是球的 Z 负方向边碰角，可能会碰到角
027             {
028                 float sina=(ballX-((int)((ballX-ballR)/UNIT_SIZE)+1)
                    *UNIT_SIZE)/ballR;//得到碰角时的角度相关值
029                 float cosa=(float)Math.sqrt(1-sina*sina);
030                 ballVX=jsSDX(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
                                //得到碰角后的速度
031                 ballVZ=-jsSDZ(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
032                 ballGX=0;
033                 ballGZ=0;
034                 if(Math.abs(ballVX)>SD_TZZ||Math.abs(ballVZ)>SD_TZZ)
                    {           //如果碰撞很大，则播放声音
035                 gameSurface.father.playSound(1, 0);
                                //播放移动声音
036                 }else if(Math.abs(ballVZ)<SD TZZ)
                                //如果速度小于调整值，则不弹起，而且速度值为零
037                 {
038                     GameSurfaceView.ballZ=i*UNIT_SIZE-ballR-MAP.
                        length*UNIT_SIZE/2;
039                     ballVZ=0;
040                     ballGZ=0;
```

```
041                }else if(Math.abs(ballVX)<SD TZZ)
                            //如果速度小于调整值，则不弹起，而且速度值为零
042                {
043                    GameSurfaceView.ballX=(1+i)*UNIT_SIZE-ballR
                       -MAP.length*UNIT_SIZE/2;
044                    ballVZ=0;
045                    ballGZ=0;
046                }
047                flag=true;
048            }
049            else if(BX>=0&&((int)((ballX+ballR)/UNIT_SIZE)>=0)&&MAP
              [i][(int)((ballX+ballR)/UNIT_SIZE)]==BKTG
050                    &&MAP[i-1][(int)((ballX+ballR)/UNIT_SIZE)]==KTG)
                       {  //如果是 Z 正方向运动时碰角了
051                float sina=(ballX-((int)((ballX+ballR)/UNIT SIZE)
                   *UNIT_SIZE))/ballR;
052                float cosa=(float)Math.sqrt(1-sina*sina);
053                ballVX=-jsSDX(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
054                ballVZ=-jsSDZ(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
055                if(Math.abs(ballVX)>SD_TZZ||Math.abs(ballVZ)>SD_TZZ){
056                    gameSurface.father.playSound(1, 0);     //播放移动声音
057                    }else{
058                        ballGX=0;
059                        ballGZ=0;
060                    }
061                flag=true;
062            }
063        }
064    }
065
066    if(BX>0)                //如果向 X 轴正方向走
067    {
068        for(int i=(int)((ballX+ballR)/UNIT_SIZE);i<=(int)
          (ballX+ballR+BX)/UNIT_SIZE);i++)
069        {//循环，假如它一下穿过几个格子，那么从第一个格子开始判断
070            if(MAP[(int)(ballZ/UNIT SIZE)][i]==BKTG&&MAP
               [(int)(ballZ/UNIT_SIZE)][i-1]==KTG){      //如果碰壁了
071
072                ballVX=-ballVX*VZ_TENUATION;           //速度置反，并调整
073                if((GameSurfaceView.ballX+ballVX*t+ballGX*t*t/2)>
074                ((i)*UNIT_SIZE-ballR-MAP[0].length*UNIT_SIZE/2))
                                          //如果已经紧贴墙壁了，那么速度为零
075                {
076                    GameSurfaceView.ballX=(i)*UNIT_SIZE-ballR-MAP[0].
                       length*UNIT_SIZE/2;
077                    ballGX=0;           //加速度和速度设置为零
078                    ballVX=0;
079                }
080                else
081                {
082                gameSurface.father.playSound(1, 0);        //播放移动声音
083                }
084                if(ballGX>0)                           //速度小于调整值则归零
085                {
086                    ballGX=0;
087                }
088                 flag=true;
089
```

```
090             }
091             else if(BZ<=0&&((int)((ballZ-ballR)/UNIT SIZE)>=0)&&
               (MAP[(int)((ballZ-ballR)/UNIT SIZE)][i]==BKTG)
092                      &&(MAP[(int)((ballZ-ballR)/UNIT SIZE)][i-1]==KTG))
                                                        //球的左边撞角
093             {
094                 float sina=(ballZ-((int)((ballZ-ballR)/UNIT_SIZE)+1)
                    *UNIT_SIZE)/ballR;//得到相关角度的正弦值和余弦值
095                 float cosa=(float)Math.sqrt(1-sina*sina);
096                 ballVX=-jsSDX(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
097                 ballVZ=jsSDZ(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
                                                        //得到碰角后的速度
098                 if(Math.abs(ballVX)>SD_TZZ||Math.abs(ballVZ)
                    >SD_TZZ){//速度达到一定大后才播放声音和震动
099                     gameSurface.father.playSound(1, 0);
                                                            //播放移动声音
100                     }else{
101                         ballGX=0;                       //速度归零
102                         ballGZ=0;
103                     }
104                 flag=true;
105             }
106             else if(BZ>=0&&((int)((ballZ+ballR)/UNIT SIZE) >=0)&&MAP
                [(int)((ballZ+ballR/UNIT SIZE)][i]==BKTG
107                     &&(MAP[(int)((ballZ+ballR)/UNIT SIZE)][i-1]==KTG)){
                                                        //如果右边碰角
108                 float sina=-(ballZ-((int)((ballZ+ballR)/UNIT_SIZE))
                   *UNIT_SIZE)/ballR;                   //得到相关值
109                 float cosa=(float)Math.sqrt(1-sina*sina);
110                 ballVX=-jsSDX(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
                                                        //得到碰角后的速度
111                 ballVZ=-jsSDZ(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
112
113                 if(Math.abs(ballVX)>SD_TZZ||Math.abs(ballVZ)>SD_TZZ)
114                 {
115                     gameSurface.father.playSound(1, 0);   //播放移动声音
116                     }else{
117                         ballGX=0;
118                         ballGZ=0;                       //否则速度归零
119                     }
120                 flag=true;
121             }
122         }
123     }
124     if(BX<0)
125     {
126         for(int i=(int)((ballX-ballR)/UNIT SIZE);i>=(int)
           ((ballX-ballR+BX)/UNIT SIZE);i--)
127         {//循环判断是否碰壁
128             if(MAP[(int)(ballZ/UNIT_SIZE)][i]==BKTG&&MAP[(int)(ballZ
                /UNIT_SIZE)][i+1]==KTG)
129             {//如果碰壁
130
131                 ballVX=-ballVX*VZ_TENUATION;        //速度置反并调整
132                 if((GameSurfaceView.ballX+ballVX*t+ballGX*t*t/2)<
133                 ((1+i)*UNIT_SIZE+ballR-MAP[0].length*UNIT_SIZE/2))
                     //如果已经紧贴墙壁了，那么速度归零
134                 {
```

```
135                 GameSurfaceView.ballX=(1+i)*UNIT_SIZE+ballR-MAP
                    [0].length*UNIT_SIZE/2;
136                 ballGX=0;                  //加速度和速度设置为零
137                 ballVX=0;
138             }
139             else
140             {
141             gameSurface.father.playSound(1, 0);   //播放移动声音
142             }
143
144         if(ballVX<0)
145         {
146             ballVX=-ballVX;
147         }
148         flag=true;
149     }
150     else if(BZ<=0&&((int)((ballZ-ballR)/UNIT_SIZE)>=0)&&
        (MAP[(int)((ballZ-ballR)/UNIT_SIZE)][i]==BKTG)
151             &&MAP[(int)((ballZ-ballR)/UNIT_SIZE)][i+1]==KTG)
                                                 //球左边撞角
152     {
153         float sina=(ballZ-((int)((ballZ-ballR)/UNIT SIZE)+1)
            *UNIT_SIZE)/ballR;
154         float cosa=(float)Math.sqrt(1-sina*sina);
                                              //得到相关的值
155         ballVX=jsSDX(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
                                                 //得到碰角后的速度
156         ballVZ=jsSDZ(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
157         if(Math.abs(ballVX)>SD TZZ||Math.abs(ballVZ)>SD TZZ)
                                                 //看是否要播放声音
158         {
159             gameSurface.father.playSound(1, 0);
                                                 //播放移动声音
160             }else{
161                 ballGX=0;
162                 ballGZ=0;
163             }
164         flag=true;
165     }
166     else if(BZ>=0&&((int)((ballZ+ballR)/UNIT SIZE)>=0)
        &&MAP[(int)((ballZ+ballR)/UNIT SIZE)][i]==BKTG
167             &&MAP[(int)((ballZ+ballR)/UNIT SIZE)][i+1]==KTG){
                                                 //如果右边碰角
168         float sina=-(ballZ-((int)((ballZ+ballR)/UNIT SIZE))
            *UNIT_SIZE)/ballR;
169         float cosa=(float)Math.sqrt(1-sina*sina);
                                              //得到相关值
170         ballVX=jsSDX(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
                                                 //得到碰角后的速度
171         ballVZ=-jsSDZ(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
172         if(Math.abs(ballVX)>SD TZZ||Math.abs(ballVZ)>SD TZZ)
                                                 //判断是否要播放声音
173         {
174             gameSurface.father.playSound(1, 0);
                                                 //播放移动声音
175             }else{
176                 ballGX=0;
177                 ballGZ=0;
178             }
```

```
179                 flag=true;
180             }
181         }
182     }
183
184     if(BZ<0)//向 Z 轴负方向上运动时
185     {
186         for(int i=(int)((ballZ-ballR)/UNIT_SIZE);i>=(int)
            ((ballZ-ballR+BZ)/UNIT_SIZE);i--)
187         {//循环看是否碰壁了
188             if(MAP[i][(int)(ballX/UNIT SIZE)]==BKTG&&MAP
                [i+1][(int)(ballX/UNIT SIZE)]==KTG){
189                 ballVZ=-ballVZ*VZ_TENUATION;            //将速度置反并调整
190                 if((GameSurfaceView.ballZ+ballVZ*t+ballGZ*t*t/2)
                    <=((1+i)*UNIT_SIZE+ballR-MAP.length*UNIT_SIZE/2))
191                 {//看调整后的速度，还是否会穿墙
192                     GameSurfaceView.ballZ=(1+i)*UNIT_SIZE+ballR-MAP.
                        length*UNIT_SIZE/2;
193                     ballVZ=0;
194                     ballGZ=0;
195                 }
196                 else
197                 {
198                 gameSurface.father.playSound(1, 0);   //播放移动声音
199                 }
200                 if(ballVZ<0)                        //如果速度还小于零，则置反
201                 {
202                     ballVZ=-ballVZ;
203                 }
204                 flag=true;
205             }
206             else if(BX<=0&&((int)((ballX-ballR)/UNIT_SIZE)>=0)&&
               (MAP[i][(int)((ballX-ballR)/UNIT_SIZE)]==BKTG)
207                     &&MAP[i+1][(int)((ballX-ballR)/UNIT_SIZE)]==KTG)
                                                                //左边碰角
208             {
209                 float sina=(ballX-((int)((ballX-ballR)/UNIT SIZE)+1)
                    *UNIT_SIZE)/ballR;//得到角度相关值
210                 float cosa=(float)Math.sqrt(1-sina*sina);
211                 ballVX=jsSDX(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
                                                            //得到碰角后的速度
212                 ballVZ=jsSDZ(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
213                 if(Math.abs(ballVX)>SD TZZ||Math.abs(ballVZ)>SD TZZ)
                                                            //判断是否要播放声音
214                 {
215                     gameSurface.father.playSound(1, 0);
                                                            //播放移动声音
216                 }
217                 else
218                 {
219                         ballGX=0;
220                         ballGZ=0;
221                 }
222                 flag=true;
223             }
224             else if(BX>=0&&((int)((ballX+ballR)/UNIT_SIZE)<MAP[0].
               length)&&MAP[i][(int)((ballX+ballR)
                /UNIT_SIZE)]==BKTG
225                     &&MAP[i+1][(int)((ballX+ballR)/UNIT_SIZE)]==KTG){
```

```
                                                    //右边碰角
226                 float sina=-(ballX-((int)((ballX+ballR)/UNIT_SIZE))
                    *UNIT_SIZE)/ballR;
227                 float cosa=(float)Math.sqrt(1-sina*sina);
                                                //得到相关值
228                 ballVX=-jsSDX(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
                                                //得到碰角后的速度
229                 ballVZ=jsSDZ(ballVX,ballVZ,cosa,sina)*VZ_TENUATION;
230                 if(Math.abs(ballVX)>SD_TZZ||Math.abs(ballVZ)>SD_TZZ)
                                                //看是否要播放声音
231                 {
232                     gameSurface.father.playSound(1, 0);
                                                //播放移动声音
233                 }
234                 else
235                 {
236                         ballGX=0;
237                         ballGZ=0;
238                 }
239                 flag=true;
240             }
241
242         }
243     }
244     return flag;
245 }
```

（6）判断小球是否进洞的原理图如下所示，首先根据小球所在的坐标计算出小球在地图数组中的行列，由于圆洞的圆心都是在每条行列线的交界点处，因此小球圆心坐标与它圆心所在图块的 4 个顶点的距离，即为小球与离它最近的 4 个圆洞的圆心距离。

获得了圆心距离之后我们可以设置这个圆心距离小于一定值就判断为小球进入圆洞，此处我们设置的这个值为小球半径的 3/2 即 ballR+ballRD。

将这些逻辑用代码实现如函数 PDJD 所示，首先获得小球圆心所在的数组行列 ballXx 和 ballZz，记住这里要将得到的值强制转换为 int，也就是舍弃小数点部分，这样才是小球圆心所在的数组行列对应的下标。然后分左上、左下、右上、右下 4 种情况对小球和圆洞的距离进行判断，进而得知小球是否掉进洞内。若小球掉进洞内，要将小球的圆心坐标设置为圆洞的圆心坐标，并将标志位 flagSY 设置为 false。

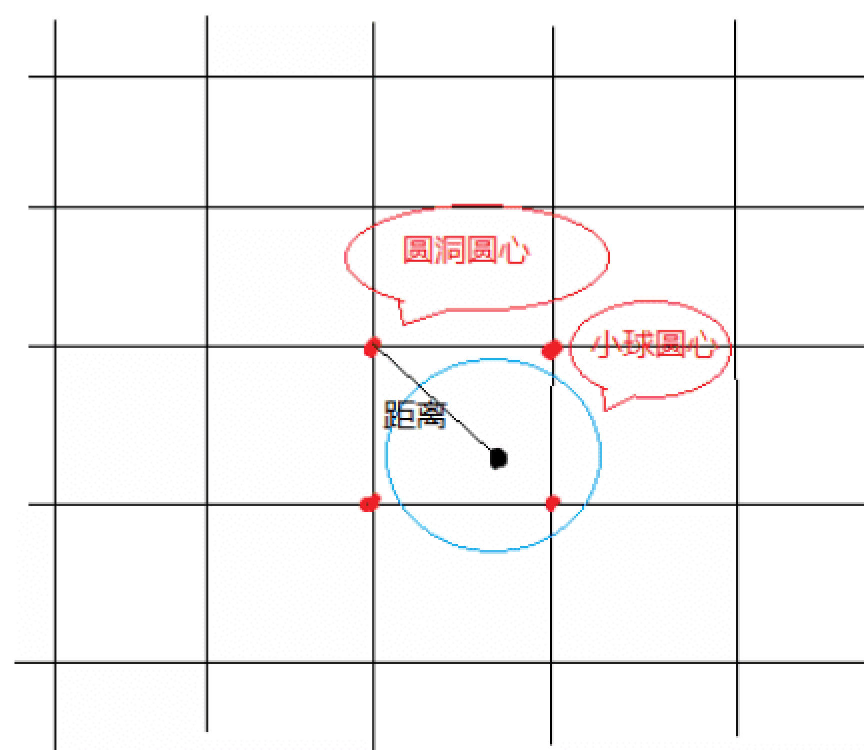


图 22.10　小球进洞判断

```
01  //判断是否进洞
02  public void PDJD()
03  {
04      //此格所在的对应的数组行列
05      ballXx=(int)((MAP[0].length*UNIT_SIZE/2+ballX)/UNIT_SIZE);
06      ballZz=(int)((MAP.length*UNIT_SIZE/2+ballZ)/UNIT_SIZE);
07      //左上的格子是洞
08      if(MAP_OBJECT[ballZz][ballXx]==1){
09              if((float)Math.sqrt(
10              (ballX-ballXx*UNIT_SIZE+MAP[0].length*UNIT_SIZE/2)
                *(ballX-ballXx*UNIT_SIZE+MAP[0].length*UNIT_SIZE/2)
11              +(ballZ-ballZz*UNIT_SIZE+MAP.length*UNIT_SIZE/2)
                *(ballZ-ballZz*UNIT_SIZE+MAP.length*UNIT_SIZE/2))
12              <ballR+ballRD                   //则判断球心是否在洞内
13              ){//掉进洞里了
14                  flagSY=false;                   //标志位
15                  ballX=ballXx*UNIT_SIZE-MAP[0].length*UNIT_SIZE/2;
                                                         //将球画到洞里
16                  ballZ=ballZz*UNIT_SIZE-MAP.length*UNIT_SIZE/2;
17                  ballY=0;
18
19      }}
20      else if(MAP_OBJECT[ballZz][ballXx+1]==1)           //左下的格子是洞
21      {
22          if((float)Math.sqrt(
23                  (ballX-(1+ballXx)*UNIT_SIZE+MAP[0].length*UNIT
                    _SIZE/2)*(ballX-(1+ballXx)*UNIT_SIZE+MAP[0].
                     length*UNIT_SIZE/2)
24                  +(ballZ-ballZz*UNIT_SIZE+MAP.length*UNIT
                     _SIZE/2)*(ballZ-ballZz*UNIT_SIZE+MAP.length
                     *UNIT_SIZE/2))
25                  <ballR+ballRD                   //则判断球心是否在洞内
26                  ){
27              flagSY=false;                       //掉进洞里了
28              ballX=(1+ballXx)*UNIT_SIZE-MAP[0].length*UNIT_SIZE/2;
29              ballZ=ballZz*UNIT_SIZE-MAP.length*UNIT_SIZE/2;
30              ballY=0;
31              ballXx=ballXx+1;
32          }
33      }
34      else if(MAP_OBJECT[ballZz+1][ballXx+1]==1){        //右下的格子是洞
35          if((float)Math.sqrt(
36                  (ballX-(1+ballXx)*UNIT_SIZE+MAP[0].length*UNIT_SIZE/2)
                    *(ballX-(1+ballXx)*UNIT_SIZE+MAP[0].length
                    *UNIT_SIZE/2)
37                  +(ballZ-(1+ballZz)*UNIT_SIZE+MAP.length*UNIT_SIZE/2)
                    *(ballZ-(1+ballZz)*UNIT_SIZE+MAP.length
                    *UNIT_SIZE/2))
38                  <ballR+ballRD                   //则判断球心是否在洞内
39                  ){
40              flagSY=false;                       //掉进洞里了
41              ballX=(1+ballXx)*UNIT_SIZE-MAP[0].length*UNIT_SIZE/2;
42              ballZ=(ballZz+1)*UNIT_SIZE-MAP.length*UNIT_SIZE/2;
43              ballY=0;
44              ballXx=ballXx+1;
45              ballZz=ballZz+1;
46          }
```

```
47      }
48      else if(MAP_OBJECT[ballZz+1][ballXx]==1){          //右上的格子是洞
49          if((float)Math.sqrt(
50                  (ballX-ballXx*UNIT_SIZE+MAP[0].length*UNIT_SIZE/2)
                    *(ballX-ballXx*UNIT_SIZE+MAP[0].length*UNIT_SIZE/2)
51                  +(ballZ-(1+ballZz)*UNIT_SIZE+MAP.length*UNIT_SIZE/2)
                    *(ballZ-(1+ballZz)*UNIT_SIZE+MAP.length*UNIT_SIZE/2))
52                  <ballR+ballRD             //则判断球心是否在洞内
53                  ){
54                      flagSY=false;             //掉进洞里了
55                      ballX=ballXx*UNIT_SIZE-MAP[0].length*UNIT_SIZE/2;
56                      ballZ=(ballZz+1)*UNIT_SIZE-MAP.length*UNIT_SIZE/2;
57                      ballY=0;
58                      ballZz=ballZz+1;
59          }
60      }
61  }
```

22.5　知 识 拓 展

本章在小球运动过程中需要用到碰撞检测，那么除了我们使用的碰撞检测方式，Android 游戏设计中还有哪些碰撞检测方式呢？其实对于碰撞检测的方式，不光 Android 很多其他语言都是大同小异，下面我们就来分别了解一下这些碰撞检测方式。

（1）地图格子划分检测

最简单的一种检测，就是把地图（或者称为场景，总之是指碰撞发生的范围）划成一个个格子，类似仙剑奇侠传这样。假设地图有 800*600px，20*20 个像素为一格。那么可以划为 40*30 个格子。地图中参与检测的对象都存储着自身所在的格子坐标，判断碰撞就显而易见了。例如可以认为两个物体在相邻格判为碰撞，或者两个物体在同一格。采用这种方式有个要求，就是地图中所有可能参与碰撞的物体都要是 20*20 像素左右大小或者是其整数倍，如房子占了 3*3 个格子，诸如此类。如果不遵守这个规则，有的物体只占了格子的一半，那么在玩家眼里这种检测就显得非常的粗糙。这种检测就像是把地图的像素点放大几十倍一样，与逐像素检测相比，效率提高了几十倍甚至上百倍。这种方式可运用于对检测要求不严格的游戏，如踩地雷的 RPG、推箱子之类的智力游戏。

（2）矩形检测

当地图中的物体不能严格按照某个块大小的整数倍来绘制时，那么就需要另想其他的方法。这种方法适用于地图中的物体近似为矩形或者虽然不是矩形，但是碰撞精度要求不高的情况下。每个物体记录一个能够将自己框住的最小矩形的左上角坐标和矩形长宽。碰撞退化为判断矩形与矩形之间是否重叠，而这仅需要 4 次比较即可得出，速度很快。但为了判断整个场景中的物体，必须取第一个物体，迭代其他所有物体进行判断，再取第二个物体，迭代除第一、第二个物体外的所有物体进行判断，以此类推。总计要进行(n-1)！次矩形判断才能准确得出场景中所有的碰撞可能。

（3）圆形检测

与上一种方法类似，区别在于用一个能够包含物体的最小圆代替了矩形。主要是考虑

到游戏中的物体外形以平滑为主，例如人物角色。而判断两个圆是否碰撞的计算也很简单，就是判断两个圆心之间的距离是否小于两个圆的半径之和。虽然球形检测在某些情况下提高了精度，但却损失了速度，因为点距离的计算需要用到平方和开方。具体相比慢多少我就不太清楚了。另外，为了计算整个地图的所有碰撞可能，也要进行(n-1)！次比较。

（4）像素检测

精确到像素级，已经不能比这更精确了，相对的，效率也是最低的。怎样判断两个物体是否碰撞呢？在过去 png 格式图片还不盛行的时候，游戏中用到的图片中的透明部分是指定用某种颜色来表示的，例如洋红色。就像电影中的绿幕蓝幕，把这些颜色的像素点当作透明点处理。而为了判断检测，需要准备一张原图像的黑白图，黑色区域表示透明。图片中的每个像素值为 0 或者 1，判断检测的时候取两张图片的黑白图，进行与运算，结果为 1（有白点重叠），则判为碰撞。但是现在有了 PNG 和 XNA，逐像素检测就相对简单一些。首先仍然需要有一个矩形框包围物体，通过矩形检测得到重叠的矩形区域可以大大减少检测的像素点数量。然后在这个区域内，取两个图片的点逐行逐列迭代，如果遇到某个点两张图片均有颜色存在，即判为碰撞。同理，进行(n-1)！次比较后得到全地图的碰撞可能。

（5）四叉树检测

准确地说这是在第 3、4、5 种方法的基础上的优化策略，或者说是第 1 种方法同后 3 种方法的组合应用。主要是针对那最后的(n-1)！次比较。方法是，像第 1 种方法一样将地图分为格子，格子的大小应该能够容纳 10 个左右的地图中最大物体，例如一个 800*600 的地图可能就划为 9 个区。同样的，每个物体要记录自己所在的区坐标以及矩形包围盒。如果该物体完全位于该区内，则只要将其与该区内的其他物体判断碰撞。如果该物体虽然位于某个区，但是小部分位于隔壁区，则额外需要迭代隔壁区的物体，相比于迭代全地图的物体，这点效率损失是可以容忍的。

有个问题，怎么知道哪些物体是跟该物体位于同一个区呢？那不是还是要迭代一遍所有物体？我们之所以称它为四叉树检测，就是因为那些区块是以四叉树的方式链接的，即得到一个区块的对象，就可以直接得到其上下左右相邻的区块的对象，而物体可以存储在所在区的一个列表中。这样就不用遍历所有物体也可以直接取出隔壁区的物体了。当地图很大的时候，四叉树的优势体现得很好。

22.6　本 章 小 结

本章介绍小 3D 重力球的设计方法，本章的游戏与之前游戏有所不同，采用 openglES 相关方法绘制图形，对于没有接触过 opengl 的读者来说，本章有些知识点可能较难理解，特别是涉及运动算法部分。由于篇幅关系，有些方面没有进行详细的介绍，请大家见谅，大家可以到网上查找相关资料，了解相关知识后再来看本章，会感觉比较轻松。